NIST Special Publication 800-56:

RECOMMENDATION ON KEY ESTABLISHMENT SCHEMES

Draft 2.0

January 2003

NIST requests that comments on this document be provided by April 3, 2003, although comments will be accepted at any time. Please send comments to **kmscomments@nist.gov**. Thanks.

Table of Contents

1. Introduction

Many U.S. Government Information Technology (IT) systems need to employ well-established cryptographic schemes to protect the integrity and confidentiality of the data that they process. Algorithms such as the Advanced Encryption Standard (AES) as defined in Federal Information Processing Standard (FIPS) 197, Triple DES as adopted in FIPS 46-3, and HMAC as defined in FIPS 198 make attractive choices for the provision of these services. These algorithms have been standardized to facilitate interoperability between systems. However, the use of these algorithms requires the establishment of shared keying material in advance. Trusted couriers may manually distribute this keying material. However, as the number of entities using a system grows, the work involved in the distribution of the keying material could grow exponentially. Therefore, it is essential to support the cryptographic algorithms used in modern U.S. Government applications with automated key establishment schemes.

2. Scope and Purpose

This Recommendation provides specifications of key establishment schemes that are appropriate for use by the U.S. Federal Government based on standards developed by the American National Standards Institute (ANSI) X9, Inc.: ANSI X9.42 *Agreement of Symmetric Keys using Discrete Logarithm Cryptography* and ANSI X9.63 *Key Agreement and Key Transport using Elliptic Curve Cryptography*. In addition, an asymmetric-key-based key transport scheme is specified as well as a symmetric-key-based key transport scheme. It is intended that this key establishment schemes Recommendation will be updated to contain key transport scheme(s) from ANSI X9.44 *Key Agreement and Key Transport using Factoring-Based Cryptography,* when they become available.

This Recommendation provides a high level description of selected schemes from ANSI X9 standards and assumes that the reader is familiar with the details and basic concepts within those standards. The implementation of these schemes, including details such as data conversion rules, arithmetic, basis, encoding rules, etc., are available in the appropriate ANSI X9 standard. When there are differences between this Recommendation and the referenced ANSI X9 standards, this key establishment schemes Recommendation **shall** have precedence for U.S. Government applications.

This Recommendation is intended to be used in conjunction with NIST Special Publication 800-57, *Guidelines for Key Management* [8]. This key establishment schemes Recommendation, the Key Management Guideline [8], and the referenced ANSI X9 standards are intended to provide sufficient information for a vendor to implement secure key establishment using asymmetric algorithms in FIPS 140-2 [1] validated modules.

3. Definitions, Symbols and Abbreviations

3.1 Definitions

Approved FIPS approved or NIST Recommended.

Bit length	The length in bits of a bit string.
Cofactor	The order of the elliptic curve group divided by the order of the prime subgroup.
Deterministic Random Bit Generator (DRBG)	An algorithm that produces a sequence of bits from an initial value called a seed. A DRBG is often called a Pseudorandom Number (or Bit) Generator.
Entity	An individual (person), organization, device, or process. "Party" is a synonym.
Ephemeral key	An ephemeral key is intended for use in exactly one instantiation of one cryptographic scheme. Contrast with static key.
Identifier	A bit string that is associated with a person, device or organization. It may be an identifying name, or may be something more abstract (e.g., a string consisting of an IP address and timestamp) depending on the application.
Initiator	The party that begins a key agreement transaction. Contrast with responder.
Key Agreement (KA)	A method of establishing keying material, whereby two parties (the initiator and the responder) contribute to the value of a shared secret from which (secret) keying material is then derived.
Key Agreement Transaction	The procedure that results in shared keying material among different parties using a key agreement scheme.
Key Confirmation (KC)	The assurance of the legitimate participants in a key establishment protocol that the parties intended to share the keying material actually possess the shared secret. The parties are called a provider and a recipient.
Key Establishment (KE)	The procedure that results in shared keying material among different parties. Key establishment may be achieved through the use of either key transport or key agreement.
Key Establishment Transaction	An instance of establishing keying material using a key establishment scheme.
Key Transport (KT)	A method of establishing a key whereby one of two parties (the sender) selects a value for their shared secret keying material and then informs the other party (the receiver) of that value.

Key Transport Transaction	The procedure that results in shared keying material between different parties using a key transport scheme.
Key Wrap	A method of encrypting keys (along with associated integrity information) that provides both confidentiality and integrity protection using a symmetric key.
Keying material	The data (e.g., a key or keys and/or other data, e.g., IVs) that is necessary to establish and maintain cryptographic keying relationships. Keying material may include keys, IVs or other information.
MacTag	Additional information that is attached to data to provide integrity protection. This information is computed on the data using a message authentication primitive. A MacTag provides data origin authentication as well as data integrity.
Message Authentication Code (MAC) algorithm	Defines a family of one-way (MAC) functions that is parameterized by a symmetric key.
Nonce	A time-varying value that has at most a negligible chance of repeating, for example, a random value that is generated anew for each instance, a timestamp, a sequence number, or some combination of these.
Owner	(1) For static keys, the entity that is authorized to use the private key, whether that entity generated the static key itself or a trusted party generated the key for the entity. (2) For ephemeral keys, the entity that generated the public/private key pair.
Party	An individual (person), organization, device, or process. "Entity" is a synonym for party.
Provider	The party in a key confirmation message exchange that provides assurance to the other party (the recipient) that they have indeed established a shared secret.
Receiver	The party that receives keying material via a key transport transaction. Contrast with sender.
Recipient	The party that receives assurance from another party, such as an assurance of the validity of a candidate public key, assurance of possession of a private key associated with a public key, or assurance of key confirmation.

Responder	The party that receives keying material from the initiator in a key agreement transaction.
Seed	An initialization value that is used (1) as input for a deterministic random number generator (DRBG) or (2) during domain parameter generation to provide assurance at a later time that the resulting domain parameters were generated randomly. For domain parameter generation, the seed may be selected arbitrarily; for a DRBG, the seed must be selected randomly, either with or without replacement.
Sender	The party that sends keying material to the receiver using a key transport transaction.
Shared keying material	The keying material that is derived by applying the key derivation function to the shared secret and other shared information.
Shared secret	A secret value that has been computed using a key agreement scheme and is used as input to a key derivation function.
Static key	A static key is intended for use for a relatively long period of time and is typically intended for use in many instances of a cryptographic key establishment scheme. Contrast with an ephemeral key.
Statistically unique	For the generation of *n*-bit quantities, the probability of repeating an explicit value is less than or equal to $\frac{1}{2^n}$.

3.2 Symbols and Abbreviations

General:

CA	Certification Authority, a trusted third party that performs services for its clients, such as creating public key certificates.
CDH	The cofactor Diffie-Hellman key agreement scheme.
DH	The (non-cofactor) Diffie-Hellman key agreement scheme.
DLC	Discrete Logarithm Cryptography, which is comprised of both Finite Field Cryptography (FFC) and Elliptic Curve Cryptography (ECC).
EC	Elliptic Curve.
ECC	Elliptic Curve Cryptography, the public key cryptographic methods using an elliptic curve, e.g., ANSI X9.63 Key Establishment [11].
FF	Finite Field.

FFC	Finite Field Cryptography, the public key cryptographic methods using a finite field, e.g., ANSI X9.42 Key Agreement [9].
IF	Integer Factorization.
IFC	Integer Factorization Cryptography, the public key cryptographic methods based on the difficultly of the integer factorization problem, e.g., ANSI X9.44 IF Key Establishment [10].
H	An Approved hash function.
MQV	The Menezes-Qu-Vanstone key agreement scheme.
$Text_1$, $Text_2$	An optional bit string that may be used during key confirmation and that is sent between the parties establishing keying material.
U	The first entity of a key establishment process, or the bit string denoting the identifier of that entity.
V	The second entity of a key establishment process, or the bit string denoting the identifier of that entity.
[*X*]	Indicates that the inclusion of string *X* is optional.
$\|\|s\|\|$	Bit length of bit string s.
$X\|\|Y$	Concatenation of two strings *X* and *Y*.
$\lceil x \rceil$	The ceiling of x; the smallest integer $\geq x$. For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$.

The following notations are consistent with those used in the ANSI X9 standards; however, it should be recognized that the notation between the standards is inconsistent (e.g., *x* and *y* are used as the private and public keys in ANSI X9.42, whereas *x* and *y* are used as the coordinates of a point in ANSI X9.63).

ANSI X9.42:

(p, q, g, [*SEED, pgenCounter*]*)*	The FFC domain parameters. *p* is the (large) prime field order. *q* is the (small) prime multiplicative subgroup order. *g* is the generator of the subgroup of order *q*. *SEED* and *pgenCounter* are used in the canonical domain parameter generation process to help ensure that *p* and *q* are generated verifiably at random.
mod *p*	The reduction modulo *p* on an integer value.
r_U, r_V	Party U or Party V's ephemeral private key.
t_U, t_V	Party U or Party V's ephemeral public key.

x_U, x_V	Party U or Party V's static private key.
y_U, y_V	Party U or Party V's static public key.
Z	A shared secret that is used to derive keying material using a key derivation function.
Z_e	An ephemeral shared secret that is computed using the Diffie-Hellman primitive.
Z_s	A static shared secret that is computed using the Diffie-Hellman primitive.

ANSI X9.63:

a, b	Field elements that define the equation of an elliptic curve.
avf(P)	The associate value of the elliptic curve point P.
$d_{e,U}$, $d_{e,V}$	Party U's and Party V's ephemeral private keys.
$d_{s,U}$, $d_{s,V}$	Party U's and Party V's static private keys.
FR	An indication of the basis used.
G	A distinguished point on an elliptic curve.
h	The cofactor, which is calculated as the order of the elliptic curve divided by the order of the point G.
n	The order of the point G.
0	The point at infinity, a special point in an elliptic curve group that serves as the (additive) identity.
q	The field size.
$Q_{e,U}$, $Q_{e,V}$	Party U's and Party V's ephemeral public keys.
$Q_{s,U}$, $Q_{s,V}$	Party U's and Party V's static public keys.
x_P	The x-coordinate of a point P.
y_P	The y-coordinate of a point P.
Z	A shared secret that is used to derive key using a key derivation function.
Z_e	An ephemeral shared secret that is computed using the Diffie-Hellman primitive.
Z_s	A static shared secret that is computed using the Diffie-Hellman primitive.

4. Key Establishment Schemes Overview

Cryptographic keying material may be electronically established between parties by using a key establishment scheme, i.e., by using either a key agreement (KA) scheme or a key transport scheme. During key agreement (where both parties contribute to the derived keying material), the keying material to be established is not sent directly; rather, information is exchanged between both parties that allows each party to derive the keying material. Key agreement schemes may use either symmetric key or asymmetric key (public key) techniques. The key agreement schemes described in this Recommendation use public key techniques. During key transport (where one party determines the keying material), wrapped (i.e., encrypted) keying material is transported from the sender, who selects that keying material and sends it to the receiver. Key transport schemes may use either symmetric key or public key techniques; both techniques are described in this Recommendation, as is the method of wrapping the keying material to be transported.

The security of the Discrete Logarithm Cryptography (DLC) schemes in this Recommendation is based on the intractability of the discrete logarithm problem. The schemes calculated over a finite field (FF) are based on ANSI X9.42. The schemes calculated using elliptic curves (EC) are based on ANSI X9.63.

This Recommendation specifies several processes that are used during key establishment (e.g., a process for generating domain parameters or a process for deriving keying material from a shared secret). In each case, an equivalent process may be used. Two processes are equivalent if, when the same values are input to each process (either as input parameters or as values made available during the process), the same output is produced.

5. Cryptographic Elements

This Recommendation assumes that the reader has, and is familiar with, ANSI X9.42 and ANSI X9.63. These standards **should** be consulted to obtain specific guidance, including data conversion rules. All calculations in an implementation **shall** be done in such a way that the user of the implementation has assurance that the arithmetic calculations are correct.

5.1 Domain Parameters

Discrete Logarithm Cryptography (DLC), that is, Finite Field Cryptography (FFC) and Elliptic Curve Cryptography (ECC), requires that the public and private key pairs be generated with respect to a particular set of domain parameters. A user of a candidate set of domain parameters **shall** have assurance of their validity prior to using them. Although domain parameters are public information, they **shall** be managed so that the correct correspondence between a given key pair and its set of domain parameters is maintained for all parties that use the key pair. Domain parameters may remain fixed for an extended time period, and one set of domain parameters may be used with multiple key establishment schemes.

Some schemes in ANSI X9.42 and X9.63 allow the set of domain parameters used and associated with static keys to be different from the set of domain parameters used and associated with ephemeral keys. For this Recommendation, however, only one set of domain parameters **shall** be used during any key establishment transaction using any such scheme (i.e., the static-

key domain parameters and the ephemeral-key domain parameters used in one scheme **shall** be the same).

5.1.1 Domain Parameter Generation

5.1.1.1 FFC Domain Parameter Generation

Domain parameters for FFC schemes are of the form (*p, q, g,*[*SEED, pgenCounter*]), where *p* is the (larger) prime field order, *q* is the (smaller) prime (multiplicative) subgroup order, *g* is a generator of the *q*-order cyclic subgroup of *GF(p)*;* and *SEED* and *pgenCounter* are values used in the canonical process of generating and validating *p* and *q.* Note that ANSI X9.42 only identifies *SEED* and *pgenCounter* as being among the domain parameters in Appendix A, but this Recommendation lists them explicitly to be consistent with ANSI X9.63.

Table 1: FFC Equivalent Strengths

Bits of security	80	112	128	192	256
Bit length of subgroup order *q*	160	224	256	384	512
Bit length of field order *p*	1024	2048	3072	8192	15360

As shown in Table 1, there are five security levels that may be chosen for use in U.S. Government applications. These five levels are given in terms of bits of security and are 80, 112, 128, 192 and 256 bits; these security levels/strengths correspond to roughly 2^{80}, 2^{112}, 2^{128}, 2^{192}, and 2^{256} operations (respectively) in order to have a reasonable expectation of breaking one key using the best of currently known attacks. This table is based on the security equivalence table in the Key Management Guideline [8], with the slight exception that the size of *p* that is associated with 192 bits of security has been rounded up to 8192 so that all sizes of *p* are multiples of 1024 bits.

In general, longer FFC keys provide more security assurance but take more time and space to use. The Key Management Guideline [8] gives guidance on selecting an appropriate security level.

For this Recommendation, the size of *p* is specified as a multiple of 1024 bits. Note that ANSI X9.42 specifies the granularity of *p* in 256-bit increments, but this Recommendation is less granular. For this Recommendation, the bit length of *q* is an exact length for a specific bit length of *p*, unlike ANSI X9.42 where the length of *q* is a minimum length that depends on the length of *p*. See [3] for routines that will utilize a stronger hash function and will specify how to generate and validate *p* and *q* in a canonical way using either probabilistic primality tests or constructive (provable) primes using the Shawe-Taylor algorithm. [3] will also specify how to calculate the generator *g*, either without transitive trust, or with transitive trust in a way to that can be validated.

5.1.1.2 ECC Domain Parameter Generation

Domain parameters for ECC schemes are of the form (*q, FR, a, b,* [*SEED*], *G, n, h*), where *q* is the field size; *FR* is an indication of the basis used; *a* and *b* are two field elements that define the

equation of the curve; *SEED* is an optional bit string if the elliptic curve was randomly generated in a verifiable fashion; *G* is a generating point, (x_G, y_G) of prime order on the curve; *n* is the order of the point *G*; and *h* is the cofactor (which is equal to the order of the curve divided by *n*). Note that the field size *q* may be either a prime *p* (where *p* is an odd prime) or 2^m, where *m* is an odd prime.

FIPS 186-3 [3] specifies recommended elliptic curves for the Federal Government that may be used as ECC domain parameters.

Table 2: ECC Equivalent Strengths

Bits of security	80	112	128	192	256
Minimum bit length of subgroup order *n*	160	224	256	384	512
Maximum value of ECC cofactor *h*	65536	65536	65536	65536	65536

As shown in Table 2, there are five security levels that may be chosen for use in U.S. Government applications. These five levels are given in terms of bits of security and are 80, 112, 128, 192 and 256 bits, these security levels/strengths correspond to roughly 2^{80}, 2^{112}, 2^{128}, 2^{192}, and 2^{256} operations (respectively) in order to have a reasonable expectation of breaking one key using the best of currently known attacks. This table is based on the security equivalence table in the Key Management Guideline [8].

In general, longer ECC keys provide more security assurance but take more time and space to use. The Key Management Guideline [8] gives guidance on selecting an appropriate security level.

All elliptic curves for use by the Federal Government **shall** have a cofactor less than or equal to 65,536 (i.e., 2^{16}). This ensures that the large subgroup is unique and ensures that using the cofactor is reasonably efficient. See the ANSI X9.62-2 ECDSA Revision draft [19] Annex A for routines that will support the generation and validation of ECC domain parameters using a stronger hash function. Note that ANSI X9.62 does not have the restriction on the maximum size of the cofactor *h*. This ANSI X9.62 draft will also specify how to calculate the generator *G* either without transitive trust, or with transitive trust in a way that can be validated.

5.1.2 Assurances of Domain Parameter Validity

Secure key establishment depends on the arithmetic validity of the set of domain parameters used by the parties. Each party **shall** have assurance of the validity of a candidate set of domain parameters. Each party **shall** obtain assurance that the candidate set of domain parameters is valid in at least one of the following three ways:

1. The party itself generates the set of domain parameters according to the specified requirements.
2. The party performs an explicit domain parameter validation as specified in:

   a. FIPS 186-3 (revision of FIPS 186-2) for FFC, including a keysize check on the primes.

b. X9.62-2 ECDSA Revision for ECC, including a keysize check on the subgroup order and a cofactor check to ensure the cofactor is 65,536 or less.

3. The party has received assurance from a trusted third party (e.g., a CA or NIST[1]) that the set of domain parameters were valid at the time that they were generated by reason of either item 1 or 2 above.

The party **shall** know which method(s) of assurance were used in order for the party to determine that the provided assurance is sufficient and appropriate to meet the application's requirements.

Note: Since SHA-1 provides 80 bits of security against collision-type attacks, it is anticipated that the use of SHA-1 for certain purposes (such as hashing a message to produce a message digest for a digital signature) will be deprecated at some time in the future. Previously validated domain parameters (that is, those that had assurance of validity before SHA-1 becomes deprecated for some purposes) are expected to continue to be considered verifiably random even when the use of SHA-1 for other purposes (such as digital signatures) is deprecated.

5.1.3 Domain Parameter Management

A particular set of domain parameters **shall** be protected against modification or substitution until the set is destroyed (if and when it is no longer needed). Each private/public key pair **shall** be correctly associated with its specific set of domain parameters (e.g., by using a public key certificate).

5.2 Private and Public Keys

5.2.1 Private/Public Key Pair Generation

Static and ephemeral key pairs are generated using the same primitive.

For the FFC schemes, generate a private key x and a public key y using the domain parameters (p, q, g, [*SEED, pgenCounter*]). Each private key **shall** be statistically unique, unpredictable, and created using an Approved random number generator. See ANSI X9.42 for details.

For the ECC schemes, generate a private key d and a public key Q using the domain parameters (q, *FR*, a, b, [*SEED*], G, n, h). Each private key **shall** be statistically unique, unpredictable, and created using an Approved random number generator. See ANSI X9.63 for details.

5.2.2 Assurances of the Arithmetic Validity of a Public Key

Secure key establishment depends on the arithmetic validity of the public key. To explain the assurance requirements, some terminology needs to be defined. The owner of a static public key is the entity that is associated with the key; this is independent of whether or not the owner generated the key pair. The recipient of a static public key is the entity that is participating in a key agreement transaction with the owner. The owner of an ephemeral public key is the entity that generated the key as part of a key agreement transaction. The recipient of an ephemeral public key is the entity that receives the key during a key agreement transaction with the owner.

[1] If using an elliptic curve from the list of NIST recommended curves in FIPS 186-2[3].

Both the owner and a recipient of a candidate public key **shall** have assurance of its arithmetic validity before using it, as specified below, and **shall** know the type of assurance provided.

5.2.2.1 Owner Assurances of Static Public Key Validity

The owner of a static public key **shall** obtain assurance of its validity in one or more of the following ways:

1. Owner Full Validation - The owner performs a successful full public key validation (see Sections 5.2.2.5 and 5.2.2.6). For example, the key generation routine may do full public key validation as part of its processing.
2. TTP Full Validation – The owner receives assurance that a trusted third party (trusted by the owner) has performed a successful full public key validation (see Sections 5.2.2.5 and 5.2.2.6).
3. Owner Generation – The owner generates the public key from the private key.
4. TTP Generation – The owner has received assurance that a trusted third party (trusted by the owner) has generated the public/private key pair and has provided the key pair to the owner.

The owner **shall** know which method(s) of assurance were used in order for the owner to determine that the provided assurance is sufficient and appropriate to meet the application's requirements. Note that the use of a TTP to generate a key pair for an owner means that the TTP must be trusted (by both the owner and any recipient) to not use the owner's private key to masquerade as the owner.

5.2.2.2 Recipient Assurances of Static Public Key Validity

The recipient of a static public key **shall** obtain assurance of its validity in one or more of the following ways:

1. Recipient Full Validation - The recipient performs a successful full public key validation (see Sections 5.2.2.5 and 5.2.2.6).
2. TTP Full Validation – The recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful full public key validation (see Sections 5.2.2.5 and 5.2.2.6).
3. TTP Generation – The recipient receivesd assurance that a trusted third party (trusted by the recipient) has generated the public/private key pair and has provided the key pair to the owner.

The recipient **shall** know which method(s) of assurance were used in order for the recipient to determine that the provided assurance is sufficient and appropriate to meet the application's requirements. Note that the use of a TTP to generate a key means that the TTP must be trusted (by both the recipient and the owner) to not use the owner's private key to masquerade as the owner.

5.2.2.3 Owner Assurances of Ephemeral Public Key Validity

The owner of an ephemeral public key **has** assurance of its validity because the owner generated the key.

5.2.2.4 Recipient Assurances of Ephemeral Public Key Validity

The recipient of an ephemeral public key **shall** obtain assurance of its validity in one or more of the following ways:

1. Recipient Full Validation - The recipient performs a successful full public key validation (see Sections 5.2.2.5 and 5.2.2.6).
2. TTP Full Validation – The recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful full public key validation (see Sections 5.2.2.5 and 5.2.2.6). For example, a trusted processor may only forward an ephemeral public key to the recipient if the public key passes a full public key validation.
3. Recipient ECC Partial Validation - If using an ECC method (only), the recipient performs a successful partial public key validation (see Section 5.2.2.7).
4. TTP ECC Partial Validation – If using an ECC method (only), the recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful partial public key validation (see Section 5.2.2.7). For example, a trusted processor may only forward an ECC ephemeral public key to the recipient if it passes a partial public key validation.

The recipient **shall** know which method of assurance was used in order for the recipient to determine that the provided assurance is sufficient and appropriate to meet the application's requirements.

5.2.2.5 FFC Full Public Key Validation Routine

candidate FFC public key to ensure that it has the unique correct representation in the correct subgroup (and therefore is also in the correct multiplicative group) of the finite field specified by the associated FFC domain parameters. FFC full public key validation does not require knowledge of the associated private key and so may be done at any time by anyone. This method **shall** be used with static and ephemeral FFC public keys when assurance of the validity of the keys is obtained by method 1 or method 2 of Sections 5.2.2.1, 5.2.2.2, and 5.2.2.4.

Input:

1. (*p, q, g* [*, seed, pgenCounter*]):A valid set of FFC domain parameters, and
2. *y*: A candidate FFC public key.

Process:

1. Verify that $2 \leq y \leq p\text{-}2$.

   (Ensure that the key has the unique correct representation and range in the field.)
2. Verify that $y^q = 1 \pmod{p}$.

(Ensure that the key has the correct order in the subgroup.)

Output: If either of the above checks fails, then output "invalid". Otherwise, output "full validation success".

5.2.2.6 ECC Full Public Key Validation Routine

ECC full public key validation refers to the process of checking all the arithmetic properties of a candidate ECC public key to ensure that it has the unique correct representation in the correct (additive) subgroup (and therefore is also in the correct EC group) specified by the associated ECC domain parameters. ECC full public key validation does not require knowledge of the associated private key and so may be done at any time by anyone. This method may be used for a static ECC public key or an ephemeral ECC public key when assurance of the validity of the key is obtained by method 1 or method 2 of Sections 5.2.2.1, 5.2.2.2, and 5.2.2.4.

Input:

1. (q, FR, a, b, [$SEED$,] G, n, h): A valid set of ECC domain parameters, and
2. $Q'=(x_Q', y_Q')$: A candidate ECC public key.

Process:

1. Verify that Q' is not the point at infinity $\mathbf{0}$.

   (Partial check of the public key for an invalid range in the EC group.)

2. 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 $q = 2^m$).

   (Ensures that each coordinate of the public key has the unique correct representation of an element in the underlying field.)

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 GF(2^m).

   (Ensures that the public key is in the correct EC group.)

4. Verify that $nQ'=\mathbf{0}$.

   (Ensures that the public key has the correct order. Along with check 1, ensures that the public key is in the correct range in the correct EC subgroup.)

Output: If any of the above checks fail, then output 'invalid'. Otherwise, output 'full validation success'.

5.2.2.7 ECC Partial Public Key Validation Routine

ECC partial public key validation refers to the process of checking some (but not all) of the arithmetic properties of a candidate ECC public key to ensure that it is in the correct group (but not necessarily the correct subgroup) specified by the associated ECC domain parameters. ECC Partial Public Key Validation omits the validation of subgroup membership, and therefore is usually faster than ECC Full Public Key Validation. ECC partial public key validation does not

require knowledge of the associated private key and so may be done at any time by anyone. This method may only be used for an ephemeral ECC public key when assurance of the validity of the key is obtained by method 3 or 4 of Section 5.2.2.4

Input:

1. (q, *FR*, a, b, [*SEED*,] G, n, h): A valid set of ECC domain parameters, and
2. $Q'=(x_Q', y_Q')$: A candidate ECC public key.

Process:

1. Verify that Q' is not the point at infinity $\mathcal{O}$.

   (Partial check of the public key for an invalid range in the EC group.)

2. 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 $q = 2^m$.

   (Ensures that each coordinate of the public key has the unique correct representation of an element in the underlying field.)

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 $GF(2^m)$.

   (Ensures that the public key is in the correct EC group.)

   (Note: Since its order is not verified, there is no check that the public key is in the EC subgroup.)

Output: If any of the above checks fail, then output 'invalid'. Otherwise, output 'partial validation success'.

5.2.3 Assurances of Possession of Private Key

Text for this section will be published in a later version of this Recommendation.

5.2.4 Key Pair Management

5.2.4.1 Common Requirements on Static and Ephemeral Key Pairs

The following are common requirements on static and ephemeral key pairs:

1. A public/private key pair **shall** be correctly associated with its corresponding specific set of domain parameters. Each key pair **shall not** be used with more than one set of domain parameters. See the Key Management Guideline [8].
2. Each private key **shall** be statistically unique, unpredictable, and created using an Approved random number generator.
3. Private keys **shall** be protected from unauthorized access, disclosure, modification or substitution. See the Key Management Guideline [8].

4. Public keys **shall** be protected from unauthorized modification or substitution. This is often accomplished by using public key certificates signed by a certificate authority (CA). See the Key Management Guideline [8].

5.2.4.2 Specific Requirements on Static Key Pairs

The specific requirements on static key pairs are as follows:

1. An entity's static key pair **shall** be generated before the generation of any ephemeral key pairs with which the static key pair will be used, this includes the entity's own ephemeral keys (if any) and the ephemeral keys of the other communicating party (if any). Note: This requirement is enforced during the generation of ephemeral keys (see Section 5.2.4.3).
2. A recipient of a static public key **shall** be assured of the association between the public key, the set of domain parameters for that key, and the entity that owns the key pair (i.e., the party with whom the recipient intends to establish a key). This assurance is often provided by verifying a public-key certificate signed by a trusted third party (e.g., a CA).
3. Static public keys **shall** be obtained in a trusted manner, e.g., from a certificate signed by a CA that the entity trusts, or directly from the public key owner, provided that the public key owner is trusted by the receiving party and can be authenticated as the source of the data that is received.
4. A static key pair may be used in more than one key establishment scheme. However, one static public/private key pair **shall not** be used for different purposes (e.g., a digital signature key pair shall not be used for key establishment or vice versa).
5. An owner and a recipient of a static public key **shall** have assurance of the validity of the public key and each **shall** know the type(s) of assurance provided. This assurance may be provided through the use of a public key certificate if the CA provides sufficient assurance of validity as part of its certification process. See Section 5.2.2.
6. An owner and a recipient of a static public key **shall** have assurance of possession of the associated private key by the claimed owner of the key pair. The owner **shall** know the type of assurance associated with the possession of his own private key; the recipient shall know the type of assurance for the possession of the private key by the other party (the claimed owner) (see Section 5.2.3). This assurance may be provided through the use of a public key certificate if the CA provides sufficient assurance of possession as part of its certification process. See Section 5.2.3.

5.2.4.3 Specific Requirements on Ephemeral Key Pairs

The specific requirements on ephemeral key pairs are as follows:

1. An ephemeral private key is intended for exactly one use, during which it is created, used in the calculation of a cryptographic key establishment primitive and then destroyed. As such, an ephemeral private key **shall** be used only once in one key establishment transaction.

2. An ephemeral key pair **should** be generated as close to its time of use as possible. Ideally, an ephemeral key pair is generated just before the ephemeral public key is transmitted.
3. If using a scheme where the other party (B) uses a static key pair, an entity (A) **shall** be assured that its ephemeral public key was transmitted strictly after the other party's (B's) static key pair was generated. This assurance can be provided by the entity (A) actually possessing a copy of the other party's (B's) static public key before generating the its own (A's) ephemeral key pair; another way to obtain this assurance is by comparing the time of ephemeral key generation with a timestamp certified by a trusted third party on the other party's (B's) static public key.
4. An ephemeral private key **shall** be destroyed immediately after the shared secret is computed.
5. A recipient of an ephemeral public key **shall** have assurance of validity of the public key and **shall** know the type(s) of assurance provided. See Section 5.2.2.

5.3 Key Derivation Function (KDF)

A key derivation function (KDF) **shall** be used to derive keying material from a shared secret. There are two Approved KDF's. The concatenation KDF is the default KDF; it **should** be used if no prior arrangement is made. The ASN.1 KDF is an optional KDF that **may** be used if both communicating parties agree upon its use. The hash function used in the KDF **shall** be Approved (see Section 5.6 for the selection of an appropriate hash function).

5.3.1 Concatenation Key Derivation Function (Default)

This section specifies the key derivation function based on concatenation. This is the default KDF.

The concatenation KDF is as follows:

Function call: kdf(*Z, OtherInput*),

where *OtherInput* is *U, V, keydatalen, hashlen*, [*SharedInfo*].

Input:

1. *Z*: A bit string that is the shared secret,
2. *U* and *V*: Bit strings that denote the identifiers of the participating parties (see notes below),
3. *keydatalen*: An integer that is the length in bits of the keying material to be generated; *keydatalen* shall be less than $hashlen \times (2^{32}-1)$,
4. *hashlen*: An integer that is the length in bits of the hash function to be used to derive the keying material, and
5. *SharedInfo*: An optional bit string that consists of data shared by parties *U* and *V*.

Process:

1. Initiate a 32-bit, big-endian bit string *counter* as 00000001_{16}.
2. $j = \lceil keydatalen / hashlen \rceil$.
3. For i=1 to j by 1, do the following:

   3.1 Compute $Hash_i = H(Z \| counter \| U \| V \| [SharedInfo])$.

   3.2 Increment *counter*.
4. Let *Hhash* be set to $Hash_j$ if ($keylen \div hashlen$) is an integer, otherwise let it be set to the ($keydatalen$-($hashlen \times (j-1)$)) leftmost bits of $Hash_j$.
5. Set $DerivedKeyingMaterial = Hash_1 \| Hash_2 \| \ldots \| Hash_{j-1} \| Hhash$.

Output: The bit string *DerivedKeyingMaterial* of *keydatalen* bits.

The shared secret **shall** be zeroized before outputting any portion of the *DerivedKeyingMaterial*; this implies that the entire *DerivedKeyingMaterial* **shall** be computed before outputting any portion of it. The derived keying material may be parsed into one or more keys or other cryptographic keying material (e.g., IVs).

Any scheme attempting to call the key derivation function for a bit string of length greater than or equal to $hashlen \times (2^{32}-1)$ bits **shall** output "invalid" and stop.

Notes:

1. The values for *U* and *V* are each an identifier (i.e., a bit string that is associated with a person, device or organization). An identifier may be an identifying name, but it is not required to be so; an identifier may be something more abstract (e.g., e an IP address and timestamp) depending on the application. The values for *U* and *V* **should** be as specific as feasible for their intended use.
2. When the scheme is such that the calculations performed by the initiator are different (see Section 6.2) than the calculations performed by the responder, then *U* **shall** be the initiator, and *V* **shall** be the responder. In a scheme where both parties do the same calculations (see Sections 6.1 and 6.3), it is up to the protocol designer to decide who serves as *U* and *V*. The protocol designer may decide that *U* is the initiator, and *V* is the responder, or the protocol may choose to select *U* based on alphabetic or some other order. The requirement is that the assignment of *U* and *V* **shall** be unambiguous.

5.3.2 ASN.1 Key Derivation Function (Optional)

This section specifies the key derivation function based on ASN.1 DER encoding. It may be used if both communicating parties agree on its use.

Keying data **shall** be calculated as follows:

Let *hashlen* denote the length of the output of the hash function chosen, and let *maxhashlen* denote the maximum length of the input to the hash function.

Function call: kdf(*Z, OtherInput*)

where *OtherInput* is *keydatalen*, *hashlen*, *OtherInfo,* and *OtherInfo* is *AlgorithmID, counter*, *PartyUInfo, PartyVInfo* [*, SuppPrivInfo*][*, SuppPubInfo*].

Input:

1. *Z*: A bit string that is the shared secret,
2. *keydatalen*: An integer that is the length in bits of the keying data to be generated. *keydatalen* shall be less than $(hashlen \times (2^{32}-1))$,
3. *OtherInfo*: A bit string specified in ASN.1 DER encoding, that consists of the following information.

   3.1 Key specification information consisting of:

   3.1.1 *AlgorithmID*: A unique object identifier that indicates the algorithm(s) for which the keying data will be used, e.g., bits 1-128 are for a 128-bit AES key and bits 129-208 are for an 80-bit HMAC key.

   3.1.2 *counter*: A 32-bit octet string with initial value 0000000116. This *counter* may be incremented during the following process.

   3.2 *PartyUInf*o: A bit string that contains public information contributed by the initiator. At a minimum, *PartyUInfo* **shall** consist of the identifier of party U; *PartyUInfo* may contain other data contributed by the initiator. See notes below.

   3.3 *PartyVInfo*: A bit string that contains public information contributed by the recipient. At a minimum, *PartyVInfo* **shall** consist of the identifier of party V; *PartyVInfo* may contain other data contributed by the recipient. See notes below.

   3.4 (Optional) *SuppPrivInfo*: A bit string that contains some additional, mutually-known private information, e.g. a shared secret symmetric key communicated through a separate channel.

   3.5 (Optional) *SuppPubInfo*: A bit string that contains some additional, mutually-known public information.

   Note that the public information referred to above is specified in the protocols that use this standard.

Actions: The key derivation function is computed as follows:

1. Let $d = \lceil keydatalen / hashlen \rceil$.
2. Initialize *counter*= 00000001_{16}.
3. For $i = 1$ to d,

   3.1 Compute $h_i = H(Z \,||\, OtherInfo)$ where h_i denotes the hash value computed using the appropriate hash function.

   3.2 Convert *counter* to an integer.

3.3 Increment *counter*.

3.4 Convert *counter* to an octet string.

4. Compute *DerivedKeyingMaterial* = leftmost *keydatalen* bits of $h_1 \| h_2 \| \ldots \| h_d$.

Output: The *DerivedKeyingMaterial* as a bit string of length *keydatalen* bits.

The shared secret **shall** be zeroized before outputting any portion of the *DerivedKeyingMaterial*; this implies that the entire *DerivedKeyingMaterial* **shall** be computed before outputting any portion of it.

The key derivation function based on ASN.1 DER encoding produces keying material that is less than ($hashlen \times (2^{32}-1)$) bits in length. It is assumed that all key derivation function calls are for bit strings that are less than ($hashlen \times (2^{32}-1)$) bits in length. Any scheme attempting to call the key derivation function using a bit string that is greater than or equal to ($hashlen \times (2^{32}-1)$) bits **shall** output "invalid" and stop. Similarly, it is assumed that all key derivation function calls do not involve hashing a bit string that is more than *maxhashlen* bits in length. Any scheme attempting to call the key derivation function on a call involving hashing a bit string that is greater than *maxhashlen* bits **shall** output "invalid" and stop.

Notes:

1. The values for *U* and *V* are each an identifier (i.e., a bit string that is associated with a person, device or organization). An identifier may be an identifying name, but it is not required to be so, instead an identifier may be something more abstract (e.g., an IP address and a timestamp) depending on the application. The values for *U* and *V* **should** be as specific as feasible for their intended use.
2. When the scheme is such that the calculations performed by the initiator are different than the calculations performed by the responder (see Section 6.2), then *U* **shall** be the initiator, and *V* **shall** be the responder. In a scheme where both parties do the same calculations (see Sections 6.1 and 6.3), it is up to the protocol designer to decide who serves as U and V. The protocol designer may decide that *U* is the initiator, and *V* is the responder, or the protocol may choose to select *U* based on alphabetic or some other order. The requirement is that the assignment of *U* and *V* **shall** be unambiguous.

5.4 Message Authentication Code (MAC) Algorithm

A Message Authentication Code (MAC) algorithm defines a family of one-way (MAC) functions that is parameterized by a symmetric key. In key establishment schemes, an entity is sometimes required to compute a *MacTag* on received or derived data using the MAC function determined by a symmetric key derived from a shared secret. The MacTag is sent to another entity in order to confirm that the shared secret was correctly computed. This Recommendation **requires** that an Approved MAC algorithm be used to compute a *MacTag*, e.g., HMAC [6].

The MAC algorithm **shall** be used to provide key confirmation, when desired, and **shall** be used to validate implementations of the key establishment schemes specified in this Recommendation.. *MacTag* computation and checking are defined in Sections 5.4.1 and 5.4.2 of this Recommendation, in Section 7.8 of ANSI X9.42 and in Section 5.7 of ANSI X9.63.

5.4.1 *MacTag* computation

The computation of the *MacTag* is represented as follows:

$$MacTag = \mathrm{MAC}_{MacKey}(MacData).$$

The *MacTag* computation **shall** be performed using an Approved MAC algorithm. In the above equation, MAC represents an Approved MAC algorithm, *MacKey* represents a symmetric key, MAC_{MacKey} represents the MAC function that is determined by *MacKey*, and *MacData* represents the data on which that function is evaluated.

5.4.2 *MacTag* Checking

To check a *MacTag* for a given *MacKey* and *MacData*, the *MacTag* is computed by the receiver using the received or derived *MacData* (as specified in Section 5.4.1) and compared with the received *MacTag*. If the two *MacTag* values are equal, then it may be inferred that the *MacKey* and *MacData* values computed by each party are equal.

5.4.3 Implementation Validation Message

For purposes of validating an implementation of the schemes in this Recommendation during an implementation validation test, the value of *MacData* **shall** be the string "Standard Test Message" followed by 16 bytes containing a 128-bit field for a nonce. The default value for this field is all zeros. Different values for this field will be specified during testing.

Note: ANSI X9.42 defines *MacData* as "ANSI X9.42 Testing Message". ANSI X9.63 does not address implementation validation at this level of detail. Note that the implementation test message used for NIST validation is a different text string from the implementation test message for ANSI X9.42 validation.

5.5 Associate Value Function (ECC MQV Only)

The associate value function is used by the ECC MQV family of key agreement schemes to compute an integer associated with an elliptic curve point. This Recommendation defines avf(*P*) to be the associate value function of a point *P* (assurance of the validity of *P* has already been obtained) as defined in Section 5.6.1 of ANSI X9.63 using the domain parameters (*q*, *FR a*, *b*, [*SEED*], *G*, *n*, *h*).

Input:

1. (*q*, *FR*, *a*, *b*, [*SEED*], *G*, *n*, *h*): Domain parameters, and
2. *P*: A point not equal to the point at infinity.

Process:

1. Convert x_P to an integer using the convention specified in Section 4.3.5 of ANSI X9.63.
2. Calculate

$$x_P' = x_P \bmod 2^{\lceil f/2 \rceil} \quad (\text{where } f = \lceil \log_2 n \rceil).$$

3. Calculate associate value function

$$\text{avf}(P) = x_P' + 2^{\lceil f/2 \rceil}.$$

Output: avf(*P*), the associate value of *P*

5.6 Cryptographic Hash Functions

An Approved hash function **shall** be used when a hash function is required (e.g., for the key derivation function or to compute a MAC when HMAC as specified in FIPS 198 is used). FIPS 180-2 [2] specifies Approved hash functions. The hash function **shall** be selected in accordance with the security level provided by the domain parameters and private/public key pairs (see Table 3).

Table 3: Hash Function Selection

Bits of security	80	112	128	192	256
Hash function	SHA1	SHA224	SHA256	SHA384	SHA512

The Approved hash functions are defined in [2] except for SHA224, which is defined as follows:

SHA224(*M*) = the leftmost 224 bits of SHA256(*M*),

where *M* is the data to be hashed.

5.7 Random Number Generation

Whenever this Recommendation requires the use of a randomly generated value (e.g., for keys or nonces), the values **shall** be generated using an Approved random number generator.

5.8 DLC Primitives

Primitives for the calculation of the shared secrets are defined in the ANSI X9.42 and X9.63 standards. A primitive is a relatively simple operation that is defined as such to facilitate implementation in hardware or in a software subroutine. Each key establishment scheme requires the use of exactly one primitive. The four primitives that **shall** be used by the schemes in Section 6 are:

1. The FFC DH primitive (Section 5.8.1.1 of this Recommendation and Section 7.5.1 in ANSI X9.42): This primitive **shall** be used by the dhHybrid1, dhEphem, dhHybridOneFlow, dhOneFlow and dhStatic schemes, which are based on finite field cryptography and the Diffie-Hellman algorithm.
2. The ECC CDH primitive (Section 5.8.1.2 of this Recommendation and called the Modified Diffie-Hellman primitive in Section 5.4.2 of ANSI X9.63): This primitive **shall** be used by the Full Unified Model, Ephemeral Unified Model, One-Pass Unified Model, One-Pass Diffie-Hellman and Static Unified Model schemes, which are based on elliptic curve cryptography and the Diffie-Hellman algorithm.
3. The FFC MQV primitive (Section 5.8.2.1 of this Recommendation): This primitive **shall** be used by the MQV2 and MQV1 schemes, which are based on finite field cryptography and the MQV algorithm.

4. The ECC MQV primitive (Section 5.8.2.2 of this Recommendation and Section 5.5 of ANSI X9.63): This primitive **shall** be used by the Full MQV and One-Pass MQV schemes, which are based on elliptic curve cryptography and the MQV algorithm.

The shared secret **shall** be used as input to a key derivation function (see Section 5.3).

5.8.1 Diffie-Hellman Primitives

5.8.1.1 Finite Field Cryptography Diffie-Hellman (FFC DH) Primitive

The shared secret *Z* is computed using the domain parameters (*p, q, g,* [*SEED, pgenCounter*]), the other party's public key and one's own private key. This primitive is used in Section 6 by the dhHybrid1, dhEphem, dhHybridOneFlow, dhOneFlow and dhStatic schemes. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either the initiator U or the responder V.

Input:

1. (*p, q, g,* [*SEED, pgenCounter*]): Domain parameters,
2. x_A : One's own private key , and
3. y_B : The other party's public key.

Process:

1. $Z = y_B^{x_A} \bmod p$
2. If *Z=1*, output "Failure".
3. Else, output *Z* .

Output: The shared secret Z or "Failure".

5.8.1.2 Elliptic Curve Cryptography Cofactor Diffie Hellman (ECC CDH) Primitive

The shared secret *Z* is computed using the domain parameters *(q, FR, a, b,* [*SEED*]*, G, n, h)*, the other party's public key, and one's own private key. This primitive is used in Section 6 by the Full Unified Model, Ephemeral Unified Model, One-Pass Unified Model, One-Pass Diffie-Hellman and Static Unified Model schemes. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either the initiator U or the responder V.

Input:

1. (*p, FR, a, b,* [*SEED*]*, G, n, h*): Domain parameters,
2. d_A : One's own private key , and
3. Q_B : The other party's public key.

Process:

1. Compute the point $P=hd_AQ_B$.

2. If *P*=O, output "Failure".
3. $Z=x_P$, where x_P is the *x*-coordinate of *P*.

Output: The shared secret *Z* or "Failure".

5.8.2 MQV Primitives

5.8.2.1 Finite Field Cryptography MQV (FFC MQV) Primitive

The shared secret *Z* is computed using the domain parameters (*p, q, g,* [*SEED, pgenCounter*]), the other party's public keys and one's own public and private keys. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either the initiator U or the responder V.

Input:

1. (*p, q, g* [, *SEED, pgenCounter*]): Domain parameters,
2. x_A : One's own static private key,
3. y_B : The other party's static public key,
4. r_A : One's own second private key,
5. t_A : One's own second public key, and
6. t_B : The other party's second public key.

Process:

1. $w = \lceil \|q\|/2 \rceil$.
2. $T_A = (t_A \bmod 2^w) + 2^w$.
3. $S_A = (r_A + T_A x_A) \bmod q$.
4. $T_B = (t_B \bmod 2^w) + 2^w$.
5. $Z = ((t_B (y_B{}^{T_B}))^{S_A}) \bmod p$.
6. If *Z*=1, output "Failure". Else, output *Z*.

Output: The shared secret Z or "Failure".

5.8.2.1.1 FFC MQV2 Form of the FFC MQV Primitive

This form of invoking the FFC MQV primitive is used in Section 6.1.1.3 by the MQV2 scheme. In this form, each party has both a static key pair and an ephemeral key pair. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either the initiator U or the responder V.

In this form, one's own second private and public pairs (input 4 and 5 in Section 5.8.2.1) are one's own ephemeral private and public keys (r_A and t_A), and the other party's second public key (input 6 in Section 5.8.2.1) is the other party's ephemeral public key (t_B).

5.8.2.1.2 FFC MQV1 Form of the FFC MQV Primitive

This form of invoking the FFC MQV primitive is used in Section 6.2.1.3 by the MQV1 scheme. In this form, the initiator has a static key pair and an ephemeral key pair, but the responder has only a static key pair. One-Pass MQV (store and forward form) is done using the MQV primitive by using the responder's static key pair as the responder's second key pair (as the responder has no ephemeral key pair).

The initiator uses the responder's static public key for the responder's second public key, i.e., when the initiator uses the algorithm in Section 5.8.2.1, input 6 becomes the other party's static public key (y_A).

The responder uses his static private key for his second private key, i.e., when the responder uses the algorithm in Section 5.8.2.1, input 4 becomes the responder's static private key x_A, and input 5 becomes the responder's static public key (y_A).

5.8.2.2 Elliptic Curve Cryptography MQV (ECC MQV) Primitive

The ECC MQV primitive is computed using the domain parameters *(q, FR, a, b,* [*SEED*], *G, n, h),* the other party's public keys, and one's own public and private keys. The ECC version of MQV uses the cofactor *h* in its calculations. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either the scheme initiator U or the scheme responder V.

Input:

1. (*q*, *FR*, *a*, *b*, [*SEED*], *G*, *n*, *h*): Domain parameters,
2. $d_{s,A}$: One's own static private key,
3. $Q_{s,B}$: The other party's static public key,
4. $d_{e,A}$: One's own second private key,
5. $Q_{e,A}$: One's own second public key, and
6. $Q_{e,B}$: The other party's second public key.

Process:

1. $\text{implicitsig}_A = (d_{e,A} + \text{avf}(Q_{e,A})d_{s,A}) \bmod n$.
2. $P = h(implicitsig_A)(Q_{e,B} + \text{avf}(Q_{e,B})Q_{s,B}))$.
3. If P = O, output "Failure".
4. $Z = x_P$, where x_P is the x-coordinate of P.

Output: Z or "Failure".

5.8.2.2.1 ECC Full MQV Form of the ECC MQV Primitive

This form of invoking the FFC MQV primitive is used in Section 6.1.1.4 by the Full MQV scheme. In this form, each party has both a static key pair and an ephemeral key pair. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either the initiator U or the responder V.

In this form one's own second key pair is one's own ephemeral key pair and the other party's second key pair is the other party's ephemeral key pair.

5.8.2.2.2 ECC One-Pass Form of the ECC MQV MQV Primitive

This form of invoking the ECC MQV primitive is used in Section 6.2.1.4 by the One-Pass MQV scheme. In this form, the initiator has a static key pair and an ephemeral key pair, but the responder has only a static key pair. One-Pass MQV (store and forward form) is done using the MQV primitive using the responder's static key pair as the responder's second key pair (as the responder has no ephemeral keys).

The initiator uses the responder's static public key as the responder's second public key. When the initiator uses the algorithm in Section 5.8.2.2, input 6 becomes the other party's static public key ($Q_{s,B}$).

The responder uses his static private key as his second private key. When the responder uses the algorithm in Section 5.8.2.2, input 4 becomes the responder's static private key $d_{s,A}$, and input 5 becomes the responder's static public key ($Q_{s,A}$).

5.9 RSA Primitives

To be added as ANSI X9.44 [10] becomes a standard.

5.10 Symmetric Key Wrapping Primitive

See the AES key wrapping specification [16].

6. Key Agreement

This Recommendation provides three **categories** of key agreement schemes (See Table 4). The classification of the categories is based on the number of ephemeral keys used by the two parties to the key agreement process, parties U and V. In category C(*i*), parties U and V have a total of *i* ephemeral key pairs. The first category, C(2), consists of schemes requiring the generation of ephemeral key pairs by both parties; a C(2) scheme is suitable for an interactive scenario. The second category, C(1), consists of schemes requiring the generation of an ephemeral key pair by only one party; a C(1) scheme is suitable for a store and forward scenario, but may also be used in an interactive scenario. The third category, C(0), consists of schemes that do not use ephemeral keys; C(0) schemes are suitable for static scenarios (e.g., public bulletin boards), but may also be used in interactive and store-and-forward scenarios.

Key confirmation may be added to any scheme if desired, see Section 8 for details on obtaining key confirmation.

Table 4: Key Agreement Scheme Categories

Category	Comment
C(2): Two ephemeral keys	Each party generates an ephemeral key pair.
C(1): One ephemeral key	Only the initiator generates an ephemeral key pair.
C(0): Zero ephemeral keys	No ephemeral keys are used.

Each category is comprised of one or more subcategories that are classified by the use of static keys by the parties (see Table 5). In subcategory C(*i,j*), parties U and V have a total of *i* ephemeral key pairs and *j* static key pairs.

Table 5: Key Agreement Scheme Subcategories

Category	Subcategory
C(2): Two ephemeral keys	C(2,2): Each party generates an ephemeral key pair and has a static key pair.
	C(2,0): Each party generates an ephemeral key pair; no static keys are used.
C(1): One ephemeral key	C(1,2): The initiator generates an ephemeral key pair and has a static key pair; the responder has only a static key pair.
	C(1,1): The initiator generates an ephemeral key pair, but has no static key pair; the responder has only a static key pair.
C(0): Zero ephemeral keys	C(0,2): Each party has only static keys.

The schemes may be further classified by whether they use FF cryptography as specified in ANSI X9.42 or EC cryptography as specified in ANSI X9.63. Note: the schemes are summarized in this Recommendation; see ANSI X9.42 or X9.63 for more details. A scheme may use either Diffie-Hellman or MQV primitives (see Section 5.8). Thus, for example, C(2,2,FFC DH) completely classifies the dhHybrid1 scheme as a scheme with two ephemeral keys and two static keys that uses finite field cryptography and a Diffie-Hellman primitive (see Table 6).

Table 6: Key Agreement Schemes

Category	Subcategory	Primitive	Scheme	Full Classification
C(2)	C(2,2)	FFC DH	dhHybrid1	C(2,2,FFC DH)
C(2)	C(2,2)	ECC CDH	(Cofactor) Full Unified Model	C(2,2,ECC CDH)
C(2)	C(2,2)	FFC MQV	MQV2	C(2,2,FFC MQV)
C(2)	C(2,2)	ECC MQV	Full MQV	C(2,2,ECC MQV)
C(2)	C(2,0)	FFC DH	dhEphem	C(2,0,FFC DH)
C(2)	C(2,0)	ECC CDH	(Cofactor) Ephemeral Unified Model	C(2,0,ECC CDH)
C(1)	C(1,2)	FFC DH	dhHybridOneFlow	C(1,2,FFC DH)
C(1)	C(1,2)	ECC CDH	(Cofactor) One-Pass Unified Model	C(1,2,ECC CDH)
C(1)	C(1,2)	FFC MQV	MQV1	C(1,2,FFC MQV)
C(1)	C(1,2)	ECC MQV	One-Pass MQV	C(1,2,ECC MQV)
C(1)	C(1,1)	FFC DH	dhOneFlow	C(1,1,FFC DH)
C(1)	C(1,1)	ECC CDH	(Cofactor) One-Pass Diffie-Hellman	C(1,1,ECC CDH)
C(0)	C(0,2)	FFC DH	dhStatic	C(0,2,FFC DH)
C(0)	C(0,2)	ECC CDH	Cofactor Static Unified Model	C(0,2,ECC CDH)

Each party in a key agreement process **shall** use the same set of domain parameters. These parameters **shall** be established prior to the initiation of the key agreement process. See Section 5.1 for a discussion of domain parameters.

A general flow diagram is provided for each subcategory of schemes. The dotted-line arrows represent the distribution of static public keys that may be distributed by the parties themselves or by a third party, such as a Certification Authority (CA). The solid-line arrows represent the distribution of ephemeral public keys that occur during the key agreement process. Note that the flow diagrams and the scheme descriptions in this Recommendation omit explicit mention of various validation checks that are required. The flow diagrams and descriptions in this Recommendation assume a successful completion of the key establishment process. The required checks are provided in the applicable ANSI standard and elsewhere in this Recommendation.

The descriptions in this section assume that an assurance of the domain parameter validity has been obtained as specified in Section 5.1.2, an assurance of static public key validity is obtained as specified in Section 5.2.2.1, and an assurance of ephemeral public key validity is obtained as specified in Section 5.2.2.2 (i.e., these processes are not mentioned in the descriptions). If these assurances are not obtained, the key establishment process **shall** be discontinued.

A description of the security attributes for each subcategory, C(*i*,*j*), is included. These sections will provide the user or developer with additional information to help make a choice as to which key establishment scheme to use. In general the attributes for each scheme within a subcategory are the same; when this is not the case, the exceptions are pointed out. See Section 6.1.1.5 specifically. These sections do not constitute an in-depth discussion of all possible security attributes of all schemes; for example, the compromise of a static private key will allow an adversary to impersonate the owner of that key, regardless of which scheme is used. For further discussion, see Annex E of ANSI X9.42 (specifically E.2.2) and Annex H of ANSI X9.63 (specifically H.4.3). Note that key confirmation may be added to any scheme and is needed in some cases to establish all the security attributes possible for a scheme.

It is important that a scheme not be chosen based only on the number of security attributes it possesses. Rather, a scheme should be selected based on how well the scheme fulfills the system requirements. For instance, in a bandwidth-constrained system, a scheme with fewer passes per-exchange might be preferable to a scheme with more passes and more security attributes.

6.1 Schemes Using Two Ephemeral Key Pairs, C(2)

In this category, each party generates an ephemeral key pair and sends the ephemeral public key to the other party. The two parties perform similar computations to derive their shared secret; however, the key derivation calculation (see Section 6.3) and the key confirmation calculation (if used - see Section 8) differ for the initiator and responder. In this situation, the scheme descriptions should be interpreted with U designating the initiator and V designating the responder.

This category consists of two subcategories that are determined by the use of static keys by the parties. In the first subcategory, each party has both static and ephemeral keys (see Section 6.1.1), while in the second subcategory, each party has only ephemeral keys (see Section 6.1.2).

6.1.1 Each Party Has a Static Key Pair and Generates an Ephemeral Key Pair: C(2,2)

For these schemes, each party (U and V) has a static key pair and generates an ephemeral key pair during the key agreement process. All key pairs **shall** be generated using the same domain parameters. Party U and party V obtain each other's static public keys, which have been generated prior to the key establishment process. Both parties generate ephemeral private/public key pairs and exchange the ephemeral public keys. Using the static and ephemeral keys, both parties generate a shared secret. The shared keying material is derived from the shared secret (see Figure 1).

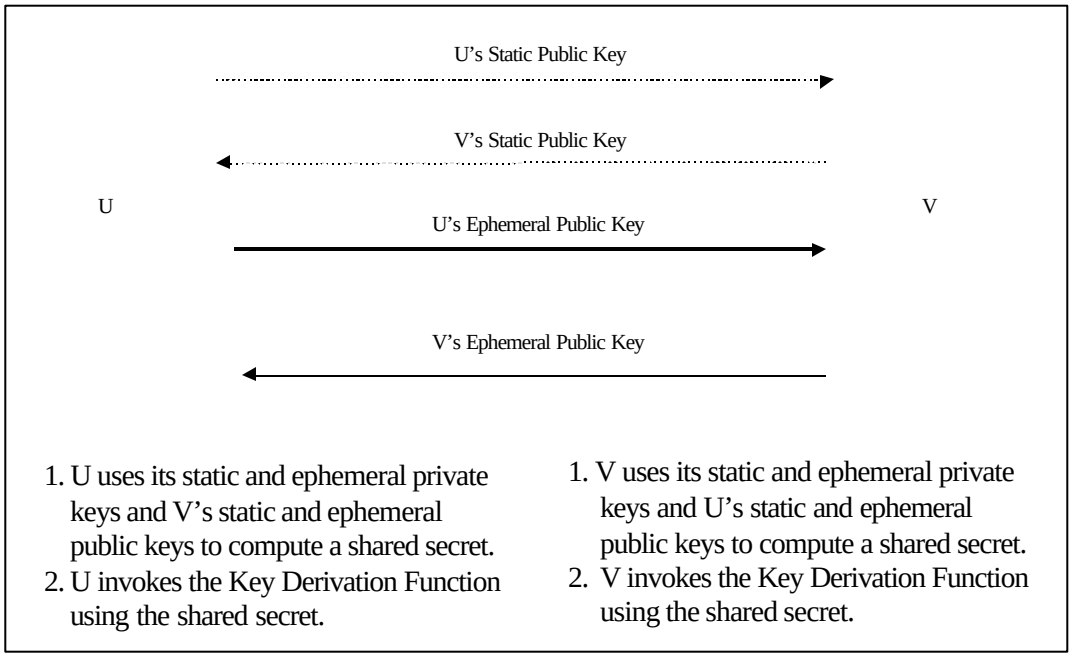

Figure 1: General Protocol When Each Party Has Both Static and Ephemeral Key Pairs

6.1.1.1 dhHybrid1, C(2,2,FFC DH)

This is a summary of the dhHybrid1 scheme from ANSI X9.42. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 8.1.4 of ANSI X9.42 for details.

In this scheme, each party has a static key pair (x, y) that was previously generated as specified in Section 5.2.1 using the same domain parameters (p, q, g, [*SEED*, *pgenCounter*]). Party U has (x_U, y_U); party V has (x_V, y_V). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, each party generates an ephemeral key pair (r, t) using the same domain parameters (p, q, g, [*SEED*, *pgenCounter*]) that were used to generate the static key pair and sends the ephemeral public key t to the other party. Party U generates (r_U, t_U) and sends

t_U to party V; party V generates (r_V, t_V) and sends t_V to party U. Each party computes the shared secret Z using the FFC DH primitive (see Section 5.8.1.1) as shown in Table 7, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 7: dhHybrid1 Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	1. Ephemeral private key r_V 2. Ephemeral public key t_V
Input	(p, q, g), x_U, y_V, r_U, t_V	(p, q, g), x_V, y_U, r_V, t_U
Computation	Compute Z_s by calling FFC DH using U's static private key and V's static public key. Compute Z_e by calling FFC DH using U's ephemeral private key and V's ephemeral public key. Compute $Z = Z_e \| Z_s$	Compute Z_s by calling FFC DH using V's static private key and U's static public key. Compute Z_e by calling FFC DH using V's ephemeral private key and U's ephemeral public key. Compute $Z = Z_e \| Z_s$
Derive Keying Material	Compute kdf(*Z,OtherInput*)	Compute kdf(*Z,OtherInput)*

6.1.1.2 Full Unified Model, C(2,2,ECC CDH)

This is a summary of the Full Unified Model scheme from ANSI X9.63. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 6.6 of ANSI X9.63 for details.

In this scheme, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 5.2.1 using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h). Party U has ($d_{s,U}$, $Q_{s,U}$); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, each party generates an ephemeral key pair (d_e, Q_e) using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h) that were used to generate the static key pair and sends the ephemeral public key Q_e to the other party. Party U generates ($d_{e,U}$, $Q_{e,U}$) and sends $Q_{e,U}$ to party V; party V generates ($d_{e,V}$, $Q_{e,V}$) and sends $Q_{e,V}$ to party U. Each party computes the shared secret Z using the ECC CDH primitive (see Section 5.8.1.2) as shown in Table 8, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 8: Full Unified Model Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	1. Ephemeral private key $d_{e,V}$ 2. Ephemeral public key $Q_{e,V}$
Input	(q, *FR* a, b, [*SEED*], G, n, h), $d_{e,U}$, $Q_{e,V}$, $d_{s,U}$, $Q_{s,V}$	(q, *FR*, a, b, [*SEED*] G, n, h), $d_{e,V}$, $Q_{e,U}$, $d_{s,V}$, $Q_{s,U}$
Computation	Compute Z_s by calling ECC CDH using U's static private key and V's static public key. Compute Z_e by calling ECC CDH using U's ephemeral private key and V's ephemeral public key. Compute $Z = Z_e \| Z_s$	Compute Z_s by calling ECC CDH using V's static private key and U's static public key. Compute Z_e by calling ECC CDH using V's ephemeral private key and U's ephemeral public key. Compute $Z = Z_e \| Z_s$
Derive Keying Material	Compute kdf(*Z*,*OtherInput*)	Compute kdf(*Z*,*OtherInput*)

6.1.1.3 MQV2, C(2,2,FFC MQV)

This is a summary of the MQV2 scheme from ANSI X9.42. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 8.2.1 of ANSI X9.42 for details.

For the MQV2 scheme, each party has a static key pair (x, y) that was previously generated as specified in Section 5.2.1 using the same domain parameters (p, q, g, [*SEED, pgenCounter*]). Party U has (x_U, y_U); party V has (x_V, y_V). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, each party generates an ephemeral key pair (r, t) using the same domain parameters (p, q, g, [*SEED*, *pgenCounter*]) that were used to generate the static key pair and sends the ephemeral public key t to the other party. Party U generates (r_U, t_U) and sends t_U to party V; party V generates (r_V, t_V) and sends t_V to party U. Each party computes the shared secret Z using the FFC MQV2 form of the FFC MQV primitive (see Section 5.8.2.1.1) as shown in Table 9, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 9: MQV2 Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	1. Ephemeral private key r_V 2. Ephemeral public key t_V
Input	(p, q, g), x_U, y_V, r_U, t_U, t_V	(p, q, g), x_V, y_U, r_V, t_V, t_U
Computation	Compute Z by calling FFC MQV using U's static private key, V's static public key, U's ephemeral private key U's ephemeral public key and V's ephemeral public key.	Compute Z by calling FFC MQV using V's static private key, U's static public key, V's ephemeral private key, V's ephemeral public key and U's ephemeral public key.
Derive Keying Material	Compute kdf(Z, *OtherInput*)	Compute kdf(Z, *OtherInput*)

6.1.1.4 Full MQV, C(2,2,ECC MQV)

This is a summary of the Full MQV scheme from ANSI X9.63. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 6.10 of ANSI X9.63 for details.

For the Full MQV scheme, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 5.2.1 using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h). Party U has ($d_{s,U}$, $Q_{s,U}$); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, each party generates an ephemeral key pair (d_e, Q_e) using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h) that were used to generate the static key pair and sends the ephemeral public key Q_e to the other party. Party U generates ($d_{e,U}$, $Q_{e,U}$) and sends $Q_{e,U}$ to party V; party V generates ($d_{e,V}$, $Q_{e,V}$) and sends $Q_{e,V}$ to party U. Each party computes the shared secret Z using the ECC Full MQV form of the ECC MQV primitive (see Section 5.8.2.2.1) as shown in Table 10, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 10: Full MQV Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$	1. Static private key $d_{s,V}$

	2. Static public key $Q_{s,U}$	2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	1. Ephemeral private key $d_{e,V}$ 2. Ephemeral public key $Q_{e,V}$
Input	(q, *FR* a, b, [*SEED*], G, n, h), $d_{e,U}$, $Q_{e,V}$, $d_{s,U}$, $Q_{e,U}$, $Q_{s,V}$	(q, *FR*, a, b, [*SEED*] G, n, h), $d_{e,V}$, $Q_{e,U}$, $d_{s,V}$, $Q_{e,V}$, $Q_{s,U}$
Computation	Compute *Z* by calling ECC MQV using U's static private key, V's static public key, U's ephemeral private key, U's ephemeral public key and V's ephemeral public key.	Compute *Z* by calling ECC MQV using V's static private key, U's static public key, V's ephemeral private key, V's ephemeral public key and U's ephemeral public key.
Derive Keying Material	Compute kdf(*Z,OtherInput*)	Compute kdf(*Z,OtherInput*)

6.1.1.5 Security Attributes of C(2,2) Schemes

These schemes provide assurance to both parties that no unintended party can compute the shared secret without the compromise of secret material.

Each party is provided with assurance that the shared secret varies from one key establishment transaction to the next. If both static and ephemeral private keys from one transaction are compromised, the shared secrets from other legitimate C(2,2) transactions are still protected by the use of different ephemeral private keys.

Key confirmation can be provided in both directions for these schemes. Upon completion of a Unilateral Key Confirmation (see Section 8.1), the recipient of the confirmation has assurance as to the identifier of the provider (through the identifier bound to the static key) as well as confirmation as to the active participation of the provider.

The MQV schemes (MQV2 and Full MQV) provide assurance to each party that if a malicious party compromises their static private key, the malicious party cannot masquerade as a third party to the party whose key was compromised. In other words, if a malicious party, E, compromises party A's static private key, then E cannot masquerade as any other party to A. The dhHybrid1 and Full Unified Model do not provide this assurance to either party.

6.1.2 Each Party Generates an Ephemeral Key Pair; No Static Keys are Used: C(2,0)

For this category, only Diffie-Hellman schemes are specified. Each party generates ephemeral key pairs with the same domain parameters. The two parties exchange ephemeral public keys and then compute the shared secret. The keying material is derived using the shared secret (see Figure 2).

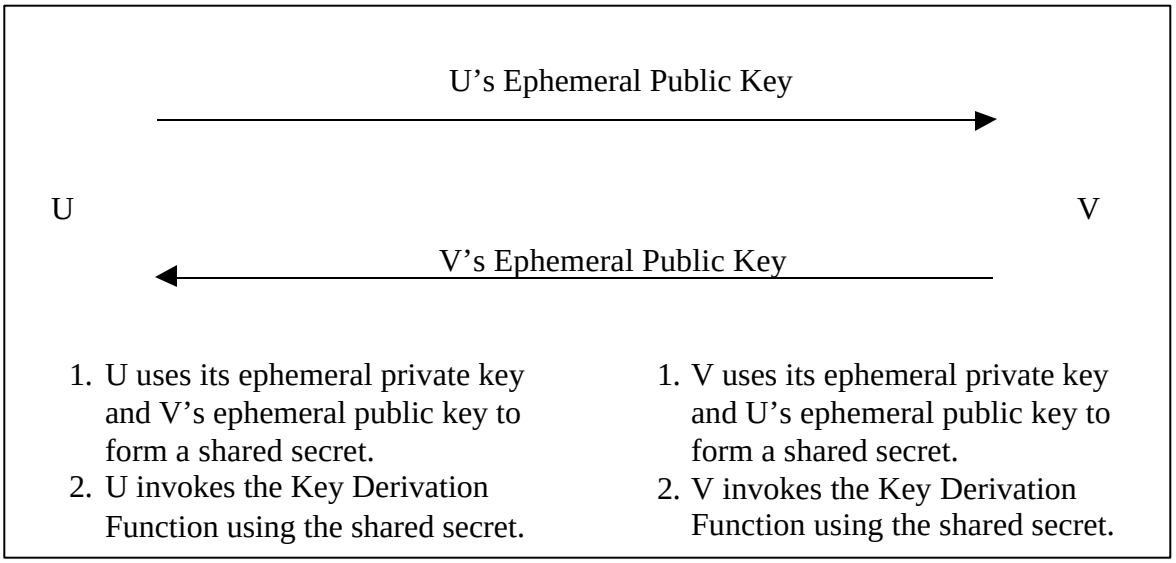

Figure 2: General Protocol When Each Party Generates Ephemeral Key Pairs; No Static Keys are Used

6.1.2.1 dhEphem, C(2,0,FFC DH)

This is a summary of the dhEphem scheme from ANSI X9.42. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 8.1.2 of ANSI X9.42 for details.

In this scheme, each party generates an ephemeral key pair (r, t) as specified in Section 5.2.1 using the same domain parameters (p, q, g, [*SEED, pgenCounter*]) and sends the ephemeral public key t to the other party. Each party computes a shared secret Z using the FFC DH primitive (see Section 5.8.1.1) as shown in Table 11. The shared keying material is computed by invoking the key derivation function using Z (see Section 5.3).

Table 11: dhEphem Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	N/A
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	1. Ephemeral private key r_V 2. Ephemeral public key t_V
Input	(p, q, g), r_U, t_V	(p, q, g), r_V, t_U
Computation	Compute Z by calling FFC DH using U's ephemeral private key and V's ephemeral public key.	Compute Z by calling FFC DH using V's ephemeral private key and U's ephemeral public key.

Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.1.2.2 Ephemeral Unified Model, C(2,0,ECC CDH)

This is a summary of the Ephemeral Unified Model scheme from ANSI X9.63. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 6.1 of ANSI X9.63 for details.

In this scheme, each party generates an ephemeral key pair (d_e, Q_e) as specified in Section 5.2.1 using the domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*) and sends the ephemeral public key Q_e to the other party. Party U generates ($d_{e,u}$, $Q_{e,u}$) and sends $Q_{e,u}$ to party V; party V generates ($d_{e,V}$, $Q_{e,V}$) and sends $Q_{e,V}$ to party U. Each party calculates a shared secret *Z* using the ECC CDH primitive (see Section 5.8.1.2) as shown in Table 12. The shared keying material is computed by invoking the key derivation function using *Z* (see Section 5.3).

Table 12: Ephemeral Unified Model Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	N/A
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	1. Ephemeral private key $d_{e,V}$ 2. Ephemeral public key $Q_{e,V}$
Input	(*q, FR a, b,* [*SEED,*] *G, n, h*), $d_{e,U}$, $Q_{e,V}$	(*q, FR, a, b,* [*SEED,*] *G, n, h*), d_{eV}, $Q_{e,U}$
Computation	Compute *Z* by calling ECC CDH using U's ephemeral private key and V's ephemeral public key.	Compute *Z* by calling ECC CDH using V's ephemeral private key and U's ephemeral public key.
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.1.2.3 Security Attributes of C(2,0) Schemes

These schemes offer no assurance to either party as to the identifier of the entity with whom they are communicating, although this can be considered a security attribute itself.

These schemes offer assurance to both parties that the current shared secret is isolated from past and future compromises of shared secrets and private keys because all cryptographic material used in the computation of the shared secret is ephemeral and is destroyed immediately after use.

Despite the fact that these schemes offer very few assurances, they are useful in many applications. For applications where, for one reason or another, there is no need to know the

identifier of the party with whom one is communicating, or where the identifier information is verified through some other method, these schemes may be appropriate.

These schemes have the property of being relatively fast to compute, due to the lack of any certificate validation. They also require no support in the form of a certificate authority. These schemes are also often used as building blocks in larger protocols where other parts of the protocol add additional security attributes.

Key Confirmation cannot be added to these schemes and, therefore, offers no additional assurances.

6.2 Schemes Using One Ephemeral Key Pair, C(1)

In this category, the parties participating in a key agreement perform different calculations to determine the shared secret, depending on whether or not they initiate the key agreement process. Let party U serve as the initiator, and party V serve as the responder. Only the initiator (party U) generates an ephemeral key pair.

This category consists of two subcategories that are determined by the possession of static key pairs by the parties. In the first subcategory, both the initiator and the responder have static key pairs, and the initiator also generates an ephemeral key pair (see Section 6.2.1). In the second subcategory, the initiator generates an ephemeral key pair, but has no static key pair; the responder has only a static key pair (see Section 6.2.2).

6.2.1 Initiator Has a Static Key Pair and Generates an Ephemeral Key Pair; Responder Has a Static Key Pair, C(1,2)

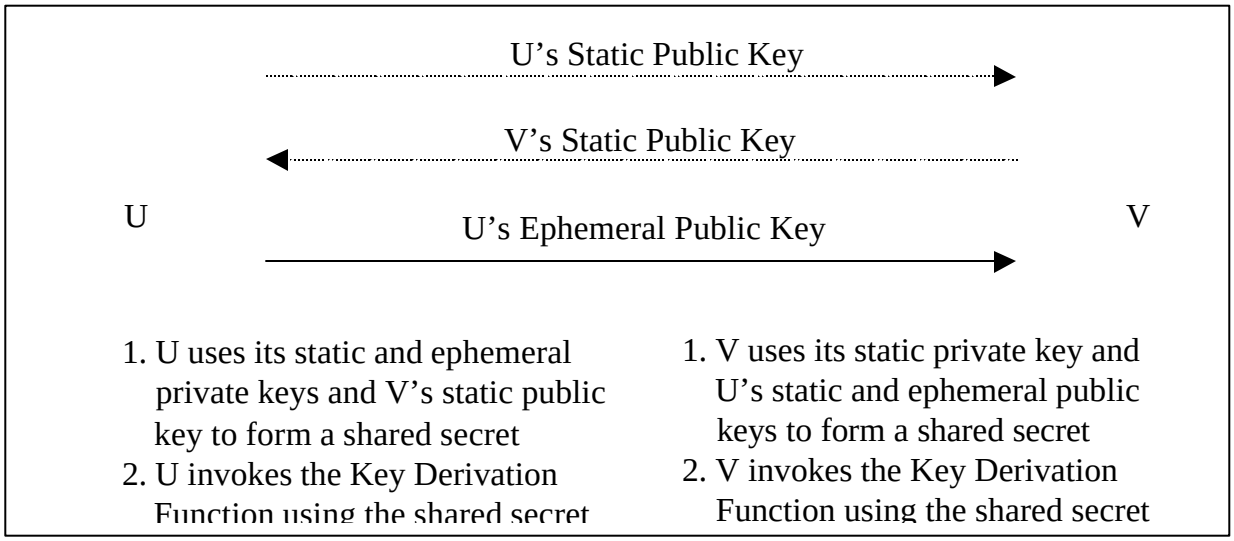

Figure 3: General Protocol When the Initiator Has Both Static and Ephemeral Key Pairs, and the Responder Has only a Static Key Pair

For these schemes, party U (the initiator) uses both static and ephemeral private/public key pairs. Party V (the responder) uses only a static private/public key pair. Party U and party V obtain

each other's static public keys in a trusted manner. Party U also sends its ephemeral public key to party V. A shared secret is generated by both parties using the available static and ephemeral keys. The shared keying material is derived using the shared secret (see Figure 3).

6.2.1.1 dhHybridOneFlow, C(1,2,FFC DH)

This is a summary of the dhHybridOneFlow scheme from ANSI X9.42. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 8.1.6 of ANSI X9.42 for details.

In this scheme, each party has a static key pair (x, y) that was previously generated as specified in Section 5.2.1 using the same domain parameters $(p, q, g, [SEED, pgenCounter])$. Party U has (x_U, y_U); party V has (x_V, y_V). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA. During the key agreement process, party U (the initiator) generates an ephemeral key pair (r_U, t_U) using the same domain parameters $(p, q, g, [SEED, pgenCounter])$ that were used to generate the static key pair and sends the ephemeral public key t_U to party V (the responder). Each party computes the shared secret Z using the FFC DH primitive (see Section 5.8.1.1) as shown in Table 13, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 13: dhHybridOneFlow Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	N/A
Input	(p, q, g), x_U, r_U, y_V	(p, q, g), x_V, y_U, t_U
Computation	Compute Z_s by calling FFC DH using U's static private key and V's static public key. Compute Z_e by calling FFC DH using U's ephemeral private key and V's static public key. Compute $Z = Z_e \| Z_s$	Compute Z_s by calling FFC DH using V's static private key and U's static public key. Compute Z_e by calling FFC DH using V's static private key and U's ephemeral public key. Compute $Z = Z_e \| Z_s$
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.2.1.2 One-Pass Unified Model, C(1,2,ECC CDH)

This is a summary of the 1-Pass Unified Model scheme from ANSI X9.63. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 6.5 of ANSI X9.63 for details.

In this scheme, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 5.2.1 using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h). Party U has (d_{sU}, Q_{sU}); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, party U (the initiator) generates an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h) that were used to generate the static key pair and sends the ephemeral public key $Q_{e,U}$ to party V (the responder). Each party computes the shared secret Z using the ECC CDH primitive (see Section 5.8.1.2) as shown in Table 14, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 14: One-Pass Unified Model Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	N/A
Input	(q, *FR*, a, b, [*SEED*], G, n, h), $d_{s,U}$, $d_{e,U}$, $Q_{s,V}$	(q, *FR*, a, b, [*SEED*], G, n, h), $d_{s,V}$, $Q_{s,U}$, $Q_{e,U}$
Computation	Compute Z_s by calling ECC CDH using U's static private key and V's static public key. Compute Z_e by calling ECC CDH using U's ephemeral private key and V's static public key. Compute $Z = Z_e \| Z_s$	Compute Z_s by calling ECC DH using V's static private key and U's static public key. Compute Z_e by calling ECC DH using V's static private key and U's ephemeral public key. Compute $Z = Z_e \| Z_s$
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.2.1.3 MQV1, C(1,2,FFC MQV)

This is a summary of the MQV1 scheme from ANSI X9.42. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 8.2.2 of ANSI X9.42 for details.

For the MQV1 scheme, each party has a static key pair (x, y) that was previously generated as specified in Section 5.2.1 using the same domain parameters (p, q, g, [*SEED, pgenCounter*]). Party U has (x_U, y_U); party V has (x_V, y_V). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, party U (the initiator) generates an ephemeral key pair (r_U, t_U) using the same domain parameters (p, q, g, [*SEED, pgenCounter*]) that were used to generate the static key pair and sends the ephemeral public key t_U to party V (the responder). Each party computes the shared secret Z using the FFC MQV1 form of the FFC MQV primitive (see Section 5.8.2.1.2) as shown in Table 15, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 15: MQV1 Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	N/A
Input	(p, q, g), x_U, y_V, r_U, t_U	(p, q, g), x_V, y_U, t_U
Computation	Compute Z by calling FFC MQV using U's static private key, V's static public key, U's ephemeral private key, U's ephemeral public key, and V's static public key (again).	Compute Z by calling FFC MQV using V's static private key, U's static public key, V's static private key (again), V's static public key and U's ephemeral public key.
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.2.1.4 One-Pass MQV, C(1,2,ECC MQV)

This is a summary of the 1-Pass MQV scheme from ANSI X9.63. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 6.9 of ANSI X9.63 for details.

For the One-Pass MQV scheme, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 5.2.1 using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h). Party U has (d_{sU}, Q_{sU}); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

Table 16:One-Pass MQV Model Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	N/A
Input	(q, *FR*, a, b, [*SEED*], G, n, h), $d_{e,U}$, $d_{s,U}$, $Q_{e,U}$, $Q_{s,V}$	(q, *FR*, a, b, [*SEED*], G, n, h), $d_{s,V}$, $Q_{s,V}$, $Q_{e,U}$, $Q_{s,U}$
Computation	Compute Z by calling ECC MQV using U's static private key, V's static public key, U's ephemeral private key, U's ephemeral public key, and V's static public key (again).	Compute Z by calling ECC MQV using V's static private key, U's static public key, V's static private key (again), V's static public key and U's ephemeral public key.
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

During the key agreement process, party U (the initiator) generates an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h) that were used to generate the static key pair and sends the ephemeral public key $Q_{e,U}$ to party V (the responder). Each party computes the shared secret Z using the ECC One-Pass MQV form of the ECC MQV primitive (see Section 5.8.2.2.2) as shown in Table 16, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

6.2.1.5 Security Attributes of C(1,2) Schemes

These schemes offer different assurances to different parties participating in the exchange. One party has both static and ephemeral keys and is called the initiator. The other party has only a static key and is called the responder.

Both parties are assured that only themselves and the other intended party can compute the shared secret. The initiator, by virtue of its ephemeral contribution, has assurance that a previous shared secret will not be reused. (Note that the addition of a per-key establishment transaction field in the [*SharedInfo*] input to the Key Derivation Function would provide assurance of non-reuse of derived keying material to any party that contributes "fresh" data to that field.)

A compromise of the static private key of the initiator does not by itself compromise past or future shared secrets (and therefore, keying material) of legitimate C(1,2) transactions, nor does the compromise of only the initiator's ephemeral private key. However, the compromise of the static private key of the responder leads to compromise of all future shared secrets where this party acts as responder. Additionally, a previous shared secret (and therefore, keying material)

computed with this party acting as responder becomes compromised if a malicious party stored the initiator's ephemeral public key.

Key confirmation can be provided in both directions. The key confirmation recipient obtains assurance of the key confirmation provider's identity and active participation.

6.2.2 Initiator Generates Only an Ephemeral Key Pair; Responder Has Only a Static Key Pair, C(1,1)

For these schemes, Party U generates an ephemeral key pair, but has no static key pair; party V has only a static key pair. Party U obtains party V's static public key in a trusted manner (e.g., from a certificate signed by a trusted CA) and sends its ephemeral public key to Party V. The parties compute a shared secret using their private keys and the other party's public key. Each party uses the shared secret to derive keying material (see Figure 4).

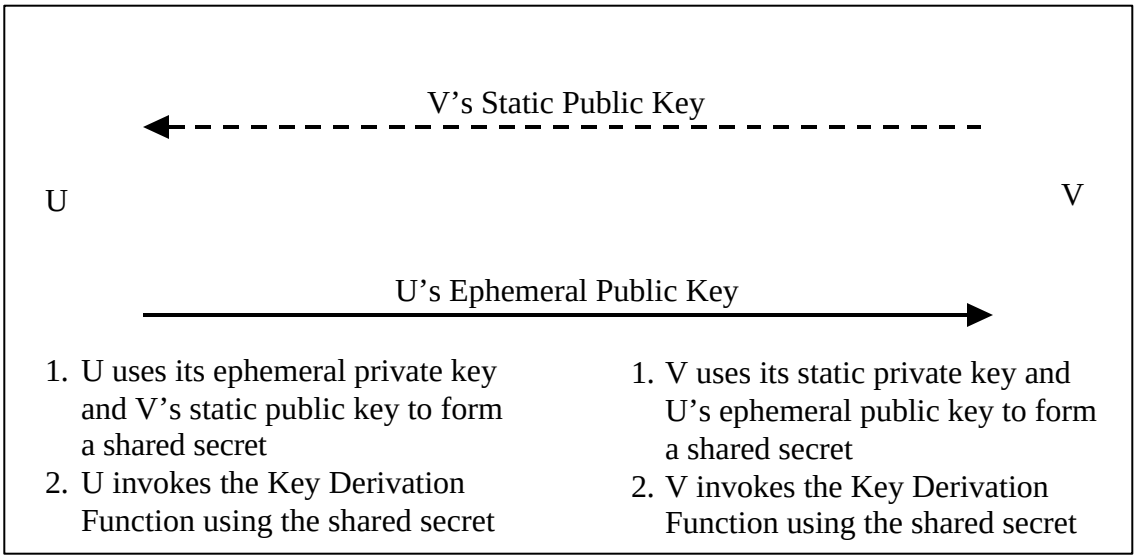

Figure 4: General Protocol When the Initiator Has Only an Ephemeral Key Pair, and the Responder Has Only a Static Key Pair

6.2.2.1 dhOneFlow, C(1,1,FFC DH)

This is a summary of the dhOneFlow scheme from ANSI X9.42. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 8.1.3 of ANSI X9.42 for details.

In this scheme, party V has a static key pair (x_V, y_V) that was previously generated as specified in Section 5.2.1 using domain parameters (*p, q, g,* [*SEED, pgenCounter*]). Party U **shall** obtain party V's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, party U (the initiator) generates an ephemeral key pair (r_U, t_U) using the same domain parameters (*p, q, g,* [*SEED, pgenCounter*]) that were used to generate party V's static key pair and sends the ephemeral public key t_U to party V (the responder). Each party computes the shared secret *Z* using the FFC DH primitive (see Section 5.8.1.1) as shown in

Table 17, and then computes the shared keying material by invoking the key derivation function using *Z* (see Section 5.3).

Table 17 : dhOneFlow Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	N/A
Input	(p, q, g), r_U, y_V	(p, q, g), x_V, t_U
Computation	Compute *Z* by calling FFC DH using U's ephemeral private key and V's static public key.	Compute Z by calling FFC DH using V's static private key and U's ephemeral public key.
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.2.2.2 One-Pass Diffie-Hellman, C(1,1,ECC CDH)

This is a summary of the 1-Pass Diffie-Hellman scheme from ANSI X9.63. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 6.2 of ANSI X9.63 for details.

In this scheme, party V has a static key pair ($d_{s,V}$, $Q_{s,V}$) that was previously generated as specified in Section 5.2.1 using domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*). Party U **shall** obtain party V's static public key ($Q_{s,V}$) in a trusted manner, e.g., from a certificate signed by a trusted CA.

During the key agreement process, party U (the initiator) generates an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using the same domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*) that were used to generate party V's static key pair and sends the ephemeral public key $Q_{e,U}$ to party V (the responder). Each party computes the shared secret *Z* using the ECC CDH primitive (see Section 5.1.8.2) as shown in Table 18, and then computes the shared keying material by invoking the key derivation function using *Z* (see Section 5.3).

Table 18: One-Pass Diffie-Hellman Model Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	N/A
Input	(*q, FR, a, b,* [*SEED*], *G, n, h*), $d_{e,U}$, $Q_{s,V}$	(*q, FR, a, b,* [*SEED*], *G, n, h*), d_{sV}, $Q_{e,U}$
Computation	Compute *Z* by calling ECC CDH using U's ephemeral private key and V's static public key.	Compute *Z* by calling ECC CDH using V's static private key and U's ephemeral public key.
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.2.2.3 Security Attributes of C(1,1) Schemes

In these schemes, one party (the initiator) has only an ephemeral key, while the other party (the responder) has only a static key. Different assurances are given to the different parties in the key establishment transaction.

These schemes provide assurance to the initiator that no unintended party can compute the shared secret without the compromise of private material. The responder has no such assurance, since the responder has no assurance about who is providing the key.

The initiator (by virtue of the ephemeral contribution) has the assurance that a previous shared secret will not be reused. The responder has no such assurance. However, the addition of a per-transaction data field to the [*SharedInfo*] field of the KDF would provide this assurance to any party that contributes "fresh" data to the per-transaction data field.

There is no assurance to either party that the security of the shared secret is isolated from compromises of private keys or shared secrets from past or future C(1,1) transactions. A compromise of the initiator's ephemeral private key compromises the shared secret for that individual transaction only. However, a compromise of the responder's static private key compromises all shared secrets resulting from future C(1,1) transactions in which that party is a responder, as well as any shared secrets resulting from past C(1,1) transactions for which a malicious party stored the ephemeral public keys.

The responder does not have any assurances as to the identifier of the initiator.

The responder may provide Key Confirmation to the initiator, giving the initiator assurance as to the identifier and active participation of the responder.

6.3 Scheme Using No Ephemeral Key Pairs, C(0,2)

In this category, each party has only static key pairs that have been generated using the same domain parameters. Each party obtains the other party's static public keys and calculates the shared secret by using their own static private key and the other party's static public key. Keying material is derived using the key derivation function and the shared secret (see Figure 5).

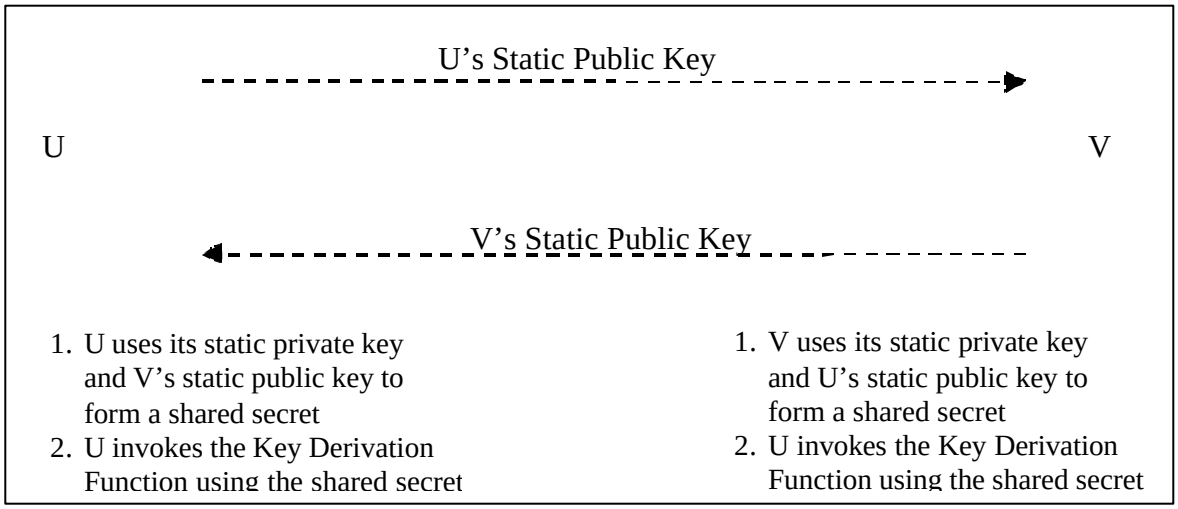

Figure 5: Each Party Has Only a Static Key Pair

6.3.1 dhStatic, C(0,2,FFC DH)

This is a summary of the dhStatic1 scheme from ANSI X9.42. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 8.1.1 of ANSI X9.42 for details.

In this scheme, each party has a static key pair (x, y) that was previously generated as specified in Section 5.2.1 using the same domain parameters (p, q, g, [*SEED, pgenCounter*]). Party U has (x_U, y_U); party V has (x_V, y_V). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

Each party computes the shared secret Z using the FFC DH primitive (see Section 5.8.1.1) as shown in Table 19, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 19: dhStatic Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	N/A	N/A

Input	(p, q, g), x_U, y_V	(p, q, g), x_V, y_U
Computation	Compute Z by calling FFC DH using U's static private key and V's static public key.	Compute Z by calling FFC DH using V's static private key and U's static public key.
Derive Keying Material	Compute kdf(*Z,OtherInput)*	Compute kdf(*Z,OtherInput)*

6.3.2 Static Unified Model, C(0,2,ECC CDH)

This is a summary of the Static Unified Model scheme from ANSI X9.63. For simplicity of presentation, some important steps (e.g., error checking and handling, and obtaining assurance of validity and possession) have been omitted. See Section 6.3 of ANSI X9.63 for details.

In this scheme, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 5.2.1 using the same domain parameters (q, *FR*, a, b, [*SEED*], G, n, h). Party U has (d_{sU}, Q_{sU}); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

Each party computes the shared secret Z using the ECC CDH primitive (see Section 5.1.8.2) as shown in Table 20, and then computes the shared keying material by invoking the key derivation function using Z (see Section 5.3).

Table 20: Static Unified Model Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	N/A	N/A
Input	(q, *FR*, a, b, [*SEED*], G, n, h), $d_{s,U}$, $Q_{s,V}$	(q, *FR*, a, b, [*SEED*], G, n, h), $d_{s,V}$, $Q_{s,U}$
Computation	Compute Z by calling ECC CDH using U's static private key and V's static public key.	Compute Z by calling ECC CDH using V's static private key and U's static public key.
Derive Keying Material	Compute kdf(*Z,OtherInput*)	Compute kdf(*Z,OtherInput*)

6.3.3 Security Attributes of C(0,2) Schemes

These schemes provide each party with assurance that the intended party and no other party can compute the shared secret.

Also, there is no variability in the shared secret computation. As described, the two participating parties will always compute the same shared secret. Variability can be provided by the addition of some per-transaction data in the [*SharedInfo*] field of the Key Derivation Function, such as a timestamp.

If an entity's private key is compromised, then all shared secrets of past and future C(0,2) transactions involving that party are compromised.

Key confirmation can be provided for these schemes to either party. Upon completion of a Unilateral Key Confirmation (Section 8.1) the recipient of the confirmation has assurance of the provider's identifier (as bound to the static key) as well as confirmation as to the active participation of the provider.

7. Key Transport

Key Transport schemes have two parties, the sender and the receiver. The sender determines the key to be transported, wraps (i.e., encrypts) the key, and sends the wrapped key to the receiver, who then unwraps (i.e., decrypts) the key. The key to be transported is wrapped using a key wrapping key (i.e., a key used to encrypt the transported key).

7.1 Symmetric-key-based Key Transport

This method uses symmetric keys to transport keying material (that is, a key or keys and/or other data to be transported) from the sender to the receiver. Both the sender and the receiver **shall** have manually established a symmetric key to be used as the key-wrapping key between the two parties. The key-wrapping key **shall** have a security in bits that is equal to or higher than the application's requirements (see the Key Management Guidelines [8]). The sender selects the keying material to be transported, wraps it using a NIST-approved key-wrapping algorithm (such as the AES key wrap algorithm [16]), and sends the wrapped keying material to the receiver. The receiver unwraps the transported keying material. The process is as follows:

1. The sender and receiver share a manually established symmetric *KeyWrappingKey*.
2. The sender selects keying material, *KeyingMaterial*, to transport to the receiver.
3. The sender calculates *WrappedKey* = KeyWrap(*KeyWrappingKey*, *KeyingMaterial*) using KeyWrap( ), an Approved key wrapping algorithm.
4. The sender sends *WrappedKey* to the receiver.
5. The receiver receives *WrappedKey* from the sender.
6. The receiver calculates *KeyingMaterial* = KeyUnwrap(*KeyWrappingKey*, *WrappedKey*) using KeyUnwrap( ), the corresponding key unwrapping algorithm.

7.2 DLC-based Key Transport

The FFC and ECC key agreement schemes in this Recommendation that employ a receiver's static key[2] may be transformed into a key transport scheme using a NIST-approved key-wrapping algorithm, such as the AES key wrap algorithm [16]). Schemes fulfilling these requirements are specified as the C(2,2), C(1,2), C(1,1) and C(0,2) schemes (see Sections 6). Key-Wrapping Key and the public keys **shall** be of an appropriate security level for the application's requirements, see the Key Management Guidelines [8]. The DLC-based key transport scheme is as follows:

1. A key agreement scheme is used to establish a shared secret between the sender and the receiver.
2. The sender obtains a *KeyWrappingKey* from the *DerivedKeyingMaterial* that is computed by applying the key derivation function to the shared secret.
3. The sender selects keying material, *KeyingMaterial*, to transport to the receiver.
4. The sender calculates *WrappedKey* = KeyWrap(*KeyWrappingKey, KeyingMaterial*) using KeyWrap( ), an Approved key wrapping algorithm.
5. The sender sends *WrappedKey* to the receiver.
6. The receiver receives *WrappedKey* from the sender.
7. The receiver obtains a *KeyWrappingKey* from the *DerivedKeyingMaterial* that is computed by applying the key derivation function to the shared secret.
8. The receiver calculates *KeyingMaterial* = KeyUnwrap(*KeyWrappingKey, WrappedKey*) using KeyUnwrap( ), the corresponding key unwrapping algorithm.

Note that if the key agreement scheme used in step 1 is such that the receiver does not contribute an ephemeral key pair to the calculation of the shared secret (that is, using either a C(1,2) or C(1,1) scheme), then steps 1 through 5 can be done by the sender without direct involvement of the receiver. This can be useful in a store-and-forward environment, such as e-mail.

7.3 IFC-based Key Transport

To be added as ANSI X9.44 becomes a standard. ANSI X9.44 is expected to specify the use of an Approved key-wrap algorithm.

8. Key Confirmation (KC)

Key confirmation is used to provide assurance to one or both participants in a key establishment process that a shared secret (e.g., *Z*, in the case of key agreement) has actually been established with the party that is believed to be the other participant. A key establishment scheme is said to provide "**unilateral key confirmation**" when it provides this assurance to only one of the participants, and the scheme is said to provide "**bilateral key confirmation**" when this assurance

[2] To prevent receiver identifier spoofing, since the sender would know the identifier of the intended receiver.

is provided to both participants (i.e., unilateral key confirmation is provided in both directions). Oftentimes, key confirmation is provided by a means outside of the key establishment scheme (for example, by decrypting an encrypted message from the other party using a key derived from the shared secret), but this is not always the case. In some cases, it may be appropriate to include the exchange of key confirmation messages within the key establishment process itself. If key confirmation is desired in one or both directions, then it may be incorporated into key establishment schemes as specified in this section.

For each unilateral direction, the party that is providing the assurance will be referred to as the key confirmation ***provider***, and the party that receives the assurance will be referred to as the key confirmation ***recipient***. Unilateral key confirmation may be incorporated into any scheme where the ***provider*** possesses a static key pair. This will provide assurance to the other party (the ***recipient***) that the individual associated with that static key pair has derived the same value for the shared secret. Bilateral key confirmation may be added to any scheme in which each party possesses a static key pair.

Table 21 provides a summary of the scheme classes for which unilateral or bilateral key confirmation is appropriate. Note that key confirmation for the C(2,0) schemes cannot be provided in the key agreement schemes, since neither party has a static key pair; if needed, key confirmation would have to be provided by some other means.

Table 21: Schemes Using Unilateral and Bilateral Key Confirmation

Scheme Class	Unilateral	Bilateral
C(2,2)	U to V and V to U	Yes
C(2,0)	No	No
C(1,2)	U to V and V to U	Yes
C(1,1)	V to U	No
C(0,2)	U to V and V to U	Yes

If key confirmation is incorporated into a scheme in which a recipient does not possess an ephemeral key pair, the recipient will need to generate a **nonce** that will be transmitted to the provider.

The process used to provide key confirmation requires string representations of the ephemeral public keys. The same notation will be used to represent these keys for schemes based on Finite Field cryptography (FFC) and elliptic curve cryptography (ECC):

$EphemPubKey_i$ = the octet string representation of a participant's ephemeral public key (t_i or $Q_{e,i}$), where i may be P to designate a provider, R to designate a recipient, or either U or V to designate the scheme initiator (party U) or the scheme responder (party V).

For FFC schemes, an ephemeral public key, t_i, is converted from a field element in F_q to an octet string by representing the field element as an integer in the interval [0, q-1], and then converting the integer to an octet string as specified in ANSI X9.42, Section 7.6.3.

For ECC, the ephemeral public key, $Q_{e,i}$, is converted from a point to an octet string as specified in ANSI X9.63, Section 4.3.6.

8.1 Unilateral Key Confirmation for Key Agreement Schemes

Unilateral key confirmation occurs when one participant in a key establishment scheme (the "provider") provides assurance to the other participant (the "recipient") that a shared secret has actually been established with the intended party. This is an optional feature for any scheme in which the provider possesses a static key pair. If the intended key confirmation recipient doe not contribute an ephemeral public key to the key establishment process, then the recipient will be required to generate a nonce to send to the key confirmation provider.Unilateral Key Confirmation may be added in either direction to the C(2,2), C(1,2) and C(0,2) schemes; it may be also be added to the C(1,1) schemes, but in one direction only: with the scheme Responder (V) is the key confirmation provider, and the scheme Initiator (U) is the key confirmation recipient (see Table 21).

To include unilateral key confirmation from a provider (who has a static key pair) to a recipient, the following steps **shall** be incorporated into the scheme. Note that the provider may be either the scheme initiator (party U) or the scheme responder (party V), as long as the provider has a static key pair, and the recipient is the other party. In the following description, *P* and *R* are the identifiers of the provider and the recipient, respectively.

1. If the recipient does not have an ephemeral key pair, then the recipient **shall** generate a nonce and send it to the provider to serve as the recipient's ephemeral data. A nonce is a time-varying value that has (at most) a negligible chance of repeating; for example, a random value generated anew for each instance, a timestamp, or a sequence number.
2. The provider computes

   $MacData_P = message\_number_P \| P \| R \| [EphemData_P] \| EphemData_R \| [Text_1]$

   where $message\_number_P$ is an integer (a one byte field) indicating the pass in the protocol that is used to transmit $MacTag_P$. The message number will be either 2 or 3. If key confirmation is unilateral, the message number will always be a 2; if key confirmation is bilateral, then the message number will be 2 for the first *MacTag* that is sent, and 3 for the second *MacTag* (see the appropriate class of schemes in Section 8.3).

   $EphemData_P$ = $EphemPubKey_P$ (if available)

   Nonce (if available and there is no $EphemPubKey_P$)

   Null (otherwise)

   $EphemData_R$ = $EphemPubKey_R$ (if available)

   Nonce (otherwise)

3. After computing the shared secret and applying the key derivation function to obtain *DerivedKeyingMaterial* (see Section 5.3), the provider parses *DerivedKeyingMaterial* into two keys, *MacKey* and *KeyData*:

$$MacKey \,||\, KeyData = DerivedKeyingMaterial$$

4. The provider computes $MacTag_P$ (see Section 5.4.1) and sends it to the recipient:

$$MacTag_P = \mathrm{MAC}_{MacKey}(MacData_P)$$

5. The recipient computes $MacData_P$, *MacKey*, *KeyData* and $MacTag_P$ in the same manner as the provider, and then compares its computed $MacTag_P$ to the value received from the provider. If the received value is equal to the derived value, then the recipient is assured that the provider has derived the same value for *MacKey* and that the provider shares the recipient's value of $MacTag_P$. The assurance of a shared value for *MacKey* provides assurance to the recipient that the provider also shares the secret value (*Z*) from which *MacKey* and *KeyData* are derived (see Section 5.3).

8.2 Bilateral Key Confirmation for Key Agreement Schemes

Bilateral key confirmation is obtained by unilateral key confirmation in both directions: from a provider V to a recipient U, and from a provider U to a recipient V. Bilateral key confirmation may be added to the C(2,2), C(1,2) and C(0,2) schemes, as shown in the relevant subsections of Section 8.3.

8.3 Incorporating Key Confirmation into Key Agreement Scheme Flow

This section provides the flow of information that is required to obtain key confirmation using the schemes of Section 6. Note that an actual communication protocol may have messages in addition to those used for key establishment.

The scheme flow descriptions in this section assume that an assurance of the domain parameter validity has been obtained as specified in Section 5.1.2, an assurance of static public key validity is obtained as specified in Section 5.2.2.1, and an assurance of ephemeral public key validity is obtained as specified in Section 5.2.2.2 (i.e., these processes are not mentioned in the flow descriptions below). If these assurances are not obtained, the key establishment process **shall** be discontinued.

The scheme flow descriptions also assume that the received MacTags are successfully verified as specified in Section 5.4.2. If the MacTags do not verify, key confirmation has not been obtained, and the key establishment process **should** be discontinued.

8.3.1 C(2,2) Scheme with Bilateral Key Confirmation

Figure 6 depicts the scheme flow for a C(2,2) scheme with bilateral key confirmation. In this method, party U, the scheme initiator, and party V, the scheme responder, assume the roles of both the provider and the recipient in order to obtain bilateral key confirmation. The successful completion of this process provides both parties with assurance that the other party has derived the same secret *Z* value and that each party has actively participated in the key establishment process.

In this scheme, each party has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

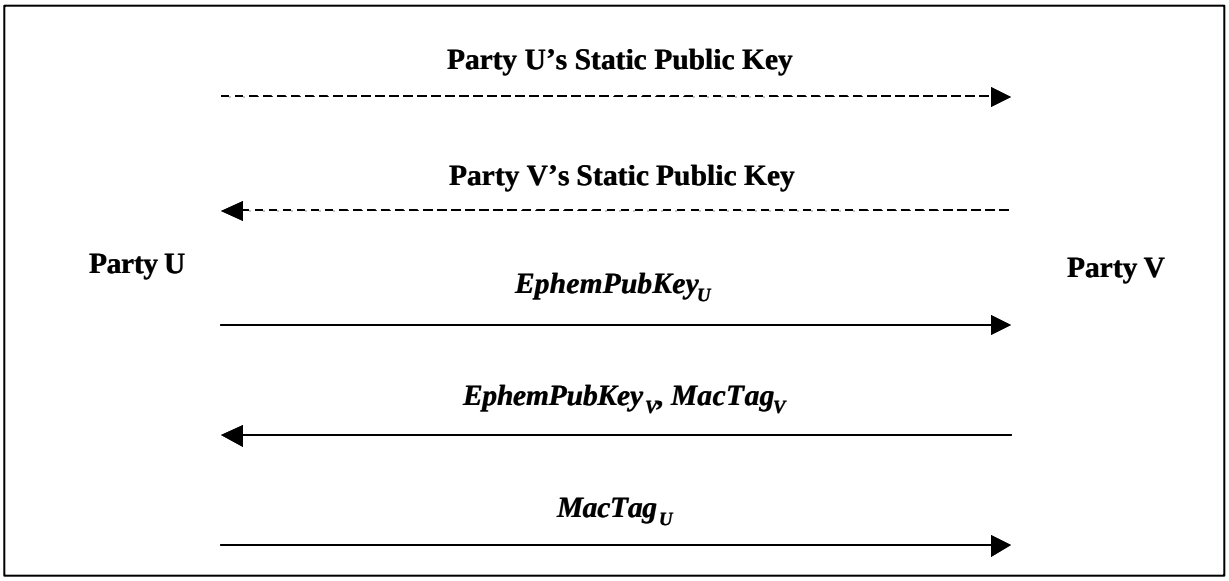

Figure 6: C(2,2) Scheme with Bilateral Key Confirmation

The flow proceeds as follows:

1. Party U generates an ephemeral key pair (see Section 5.2.1) and sends the ephemeral public key ($EphemPubKey_U$) to party V in the first message of the key establishment process.
2. Upon receiving party U's ephemeral public key, party V generates an ephemeral key pair (see Section 5.2.1) and a MacTag, and sends the ephemeral public key ($EphemPubKey_V$) and the MacTag ($MacTag_V$) to party U. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData* and sent to party V.

   $$MacData_V = 02 \| V \| U \| EphemPubKey_V \| EphemPubKey_U \| [Text_1]$$

3. Upon receiving party V's ephemeral public key and the MacTag, party U verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party U has assurance that party V has derived the same secret Z value as party U, and that party V is actively participating in the key establishment process.
4. Party U then generates a MacTag ($MacTag_U$) as specified in Sections 5.4.1 and 8.1, and sends the MacTag to party V. The *MacData* for the MacTag is:

   $$MacData_U = 03 \| U \| V \| EphemPubKey_U \| EphemPubKey_V \| [Text_2]$$

5. Upon receiving $MacTag_U$, party V verifies the MacTag. If the received and computed MacTags have the same value, then party V has assurance that party U has derived the

same secret *Z* value as party V, and that party U is actively participating in the key establishment process.

8.3.2 C(2,2) Scheme with Unilateral Key Confirmation

Figure 7 depicts the scheme flow for a C(2,2) scheme with unilateral key confirmation. In a C(2,2) scheme, party U and party V each have static key pairs, which allows either party to assume the role of the provider in a key confirmation process. Each party also has an ephemeral key pair, which allows them to assume the role of a key confirmation recipient. Therefore, either party may assume the role of provider or recipient using this key confirmation method.

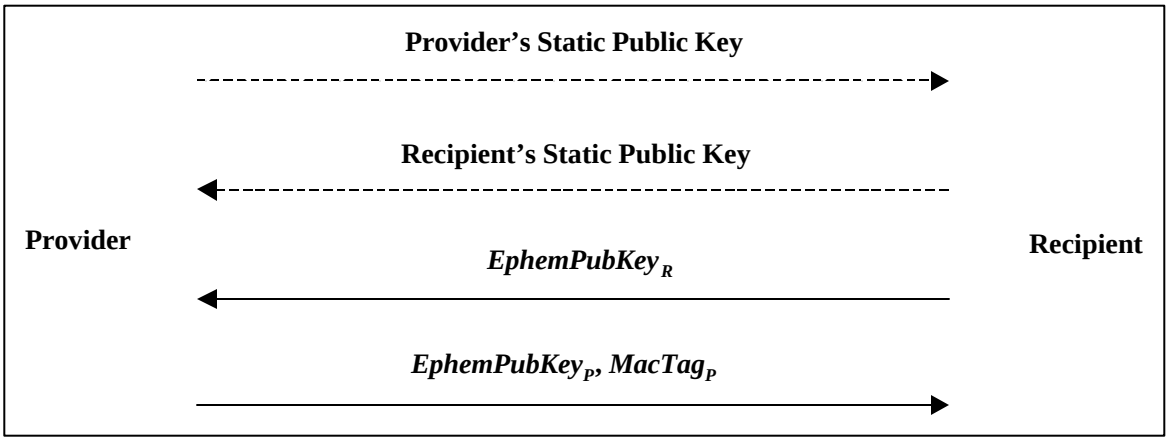

Figure 7: C(2,2) Scheme with Unilateral Key Confirmation

The successful completion of the key confirmation process assures the recipient that the provider has derived the same secret *Z* value as the recipient, and that the provider has actively participated in the key establishment process.

In this scheme, each party has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

The flow proceeds as follows:

1. The recipient generates an ephemeral key pair (see Section 5.2.1) and sends the ephemeral public key ($EphemPubKey_R$) to the provider in the first message of the key establishment process.
2. Upon receiving the ephemeral public key, the provider generates an ephemeral key pair and a MacTag, and sends the ephemeral public key ($EphemPubKey_P$) and the MacTag ($MacTag_P$) to the recipient. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData*:

   $$MacData_p = 02 \| P \| R \| EphemPubKey_P \| EphemPubKey_R \| [Text_1]$$

3. Upon receiving $MacTag_p$, the recipient verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then the recipient has assurance

that the provider has derived the same secret *Z* value as the recipient, and that the provider is actively participating in the key establishment process.

8.3.3 C(1,2) Scheme with Bilateral Key Confirmation

Figure 8 depicts the scheme flow for a C(1,2) scheme with bilateral key confirmation. In this method, party U, the scheme initiator, and party V, the scheme responder, assume the roles of both the provider and recipient in order to obtain bilateral key confirmation. However, party V does not have an ephemeral key pair to send to party U; therefore, party V **shall** send a nonce to party U in order to obtain key confirmation assurance from party U.

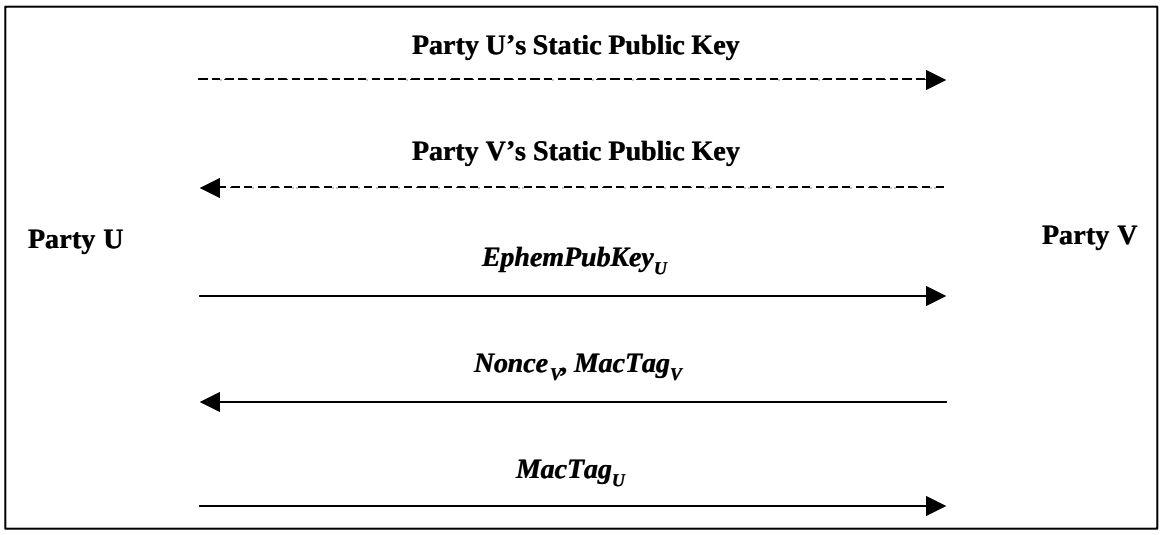

Figure 8: C(1,2) Scheme with Bilateral Key Confirmation

The successful completion of this process provides both parties with assurance that the other party has derived the same secret *Z* value and that each party has actively[3] participated in the key establishment process.

In this scheme, each party has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

The flow proceeds as follows:

1. Party U generates an ephemeral key pair (see Section 5.2.1) and sends the ephemeral public key ($EphemPubKey_U$) to party V in the first message of the key establishment process.

[3] Party V obtains assurance of active participation by Party U only if the number of unpredictable bits of the nonce sent by Party V to Party U is equal to or greater than the desired security level. For example, if 80 bits of security is desired, party V obtains assurance of active participation by party U only if the number of unpredictable bits in the nonce sent by party V to party U is equal to or greater than 80.

2. Upon receiving party U's ephemeral public key, party V generates a nonce ($Nonce_V$) and a MacTag ($MacTag_V$) and sends the nonce and the MacTag to party U. The MacTag is generated as specified in Sections 5.4.2 and 8.1 using the following as the *MacData*:

$$MacData_V = 02 \| V \| U \| Nonce_V \| EphemPubKey_U \| [Text_1]$$

3. Upon receiving party V's nonce and the MacTag, party U verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party U has received assurance that party V has derived the same secret *Z* value as party U and that party V is actively participating in the key establishment process.

4. Party U then generates a MacTag ($MacTag_U$) as specified in Sections 5.4.1 and 8.1 and sends the MacTag to party V. The *MacData* for the MacTag is:

$$MacData_U = 03 \| U \| V \| EphemPubKey_U \| Nonce_V \| [Text_2]$$

5. Upon receiving $MacTag_U$, party V verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party V has assurance that party U has derived the same secret *Z* value as party V, and that party U is actively participating in the key establishment process.

8.3.4 C(1,2) Scheme with Unilateral Key Confirmation from the Initiator to the Responder

Figure 9 depicts the scheme flow for a C(1,2) scheme with unilateral key confirmation from the initiator (party U) to the responder (party V). In a C(1,2) scheme, party U and party V each have static key pairs. Therefore, either party may assume the role of the provider in a key confirmation process. However, only one party has an ephemeral key pair. This section specifies a method for providing key confirmation to party V, who does not have an ephemeral key pair. In order for party V to obtain key confirmation from party U, party V **shall** send a nonce to party U. Section 8.3.5 specifies a method for providing key confirmation to party U, who has an ephemeral key pair.

This method of key confirmation may be used to provide party V (who is the key confirmation recipient for this method) with assurance that party U (who is the provider) has derived the same secret *Z* value as party V, and that party U has actively[4] participated in the key establishment process.

In this scheme, each party has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

[4] Party V obtains assurance of active participation by Party U only if the number of unpredictable bits of the nonce sent by Party V to Party U is equal to or greater than the desired security level.

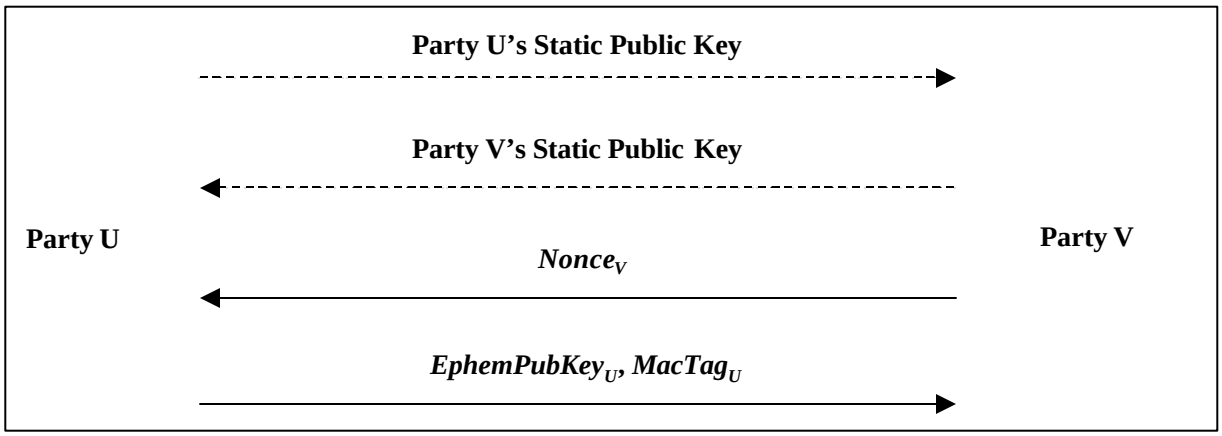

Figure 9: C(1,2) Scheme with Unilateral Key Confirmation from Party U to Party V

The flow proceeds as follows:

1. Party V generates a nonce ($Nonce_V$) and sends it to party U in the first message of the key establishment process.
2. Upon receiving party V's nonce, party U generates an ephemeral key pair (see Section 5.2.1) and a MacTag and sends the ephemeral public key ($EphemPubKey_U$) and the MacTag ($MacTag_U$) to party V. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData*:

$$MacData_U = 02 \parallel U \parallel V \parallel EphemPubKey_U \parallel Nonce_V \parallel [Text_1]$$

3. Upon receiving $MacTag_U$, party V verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party V has assurance that party U has derived the same secret *Z* value as party V, and that party U is actively participating in the key establishment process.

8.3.5 C(1,2) Scheme with Unilateral Key Confirmation from the Responder to the Initiator

Figure 10 depicts the scheme flow for a C(1,2) scheme with unilateral key confirmation from the responder (party V) to the initiator (party U). In a C(1,2) scheme, party U and party V each have static key pairs. Therefore, either party may assume the role of the provider in a key confirmation process. However, only one party has an ephemeral key pair. This section specifies a method for providing key confirmation to party U, who has an ephemeral key pair. Section 8.3.4 specifies a method for providing key confirmation to party V, who does not have an ephemeral key pair.

This method of key confirmation may be used to provide party U (who is the key confirmation recipient for this method) with assurance that party V (who is the provider) has derived the same secret *Z* value as party U, and that party V has actively participated in the key establishment process.

In this scheme, each party has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

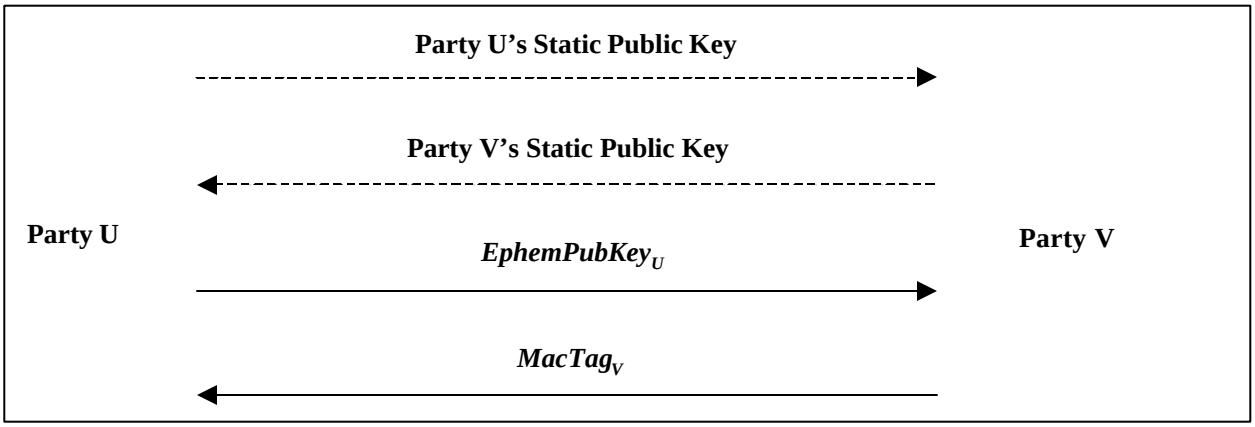

Figure 10: C(1,2) Scheme with Unilateral Key Confirmation from Party V to Party U

The flow proceeds as follows:

1. Party U generates an ephemeral key pair (see Section 5.2.1) and sends the ephemeral public key ($EphemPubKey_U$) to party V in the first message of the key establishment process.
2. Upon receiving party U's ephemeral public key, party V generates a MacTag ($MacTag_V$) and sends it to party U. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData*:

   $$MacData_V = 02 \,||\, V \,||\, U \,||\, Null \,||\, EphemPubKey_U \,||\, [Text_1]$$

   where *Null* is a null (empty) string.
3. Upon receiving $MacTag_V$, party U verifies MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party U has assurance that party V has derived the same secret *Z* value as party U, and that party V is actively participating in the key establishment process.

8.3.6 C(1,1) Scheme with Unilateral Key Confirmation from the Responder to the Initiator

Figure 11 depicts the scheme flow for a C(1,1) scheme with unilateral key confirmation from the responder (party V) to the initiator (party U). In a C(1,1) scheme, party U has an ephemeral key pair, but no static key pair. Therefore, party U cannot assume the role of the provider in a key confirmation process; but party U can be the recipient in the process. Party V has a static key pair and, therefore, can assume the role of the key confirmation provider.

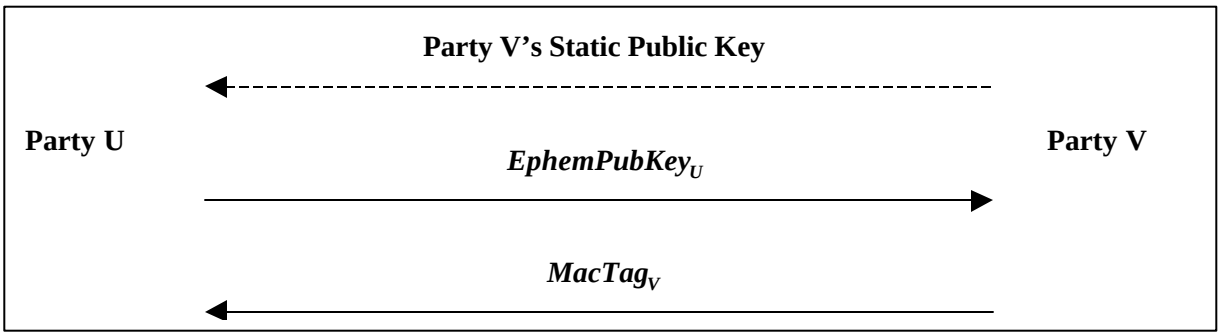

Figure 11: C(1,1) Scheme with Unilateral Key Confirmation from Party V to Party U

This method of key confirmation may be used to provide party U (who is the key confirmation recipient for this method) with assurance that party V (who is the provider) has derived the same secret *Z* value as party U, and that the party V has actively participated in the key establishment process.

In this scheme, party V has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Party U **shall** obtain party V's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

The flow proceeds as follows:

1. Party U generates an ephemeral key pair (see Section 5.2.1) and sends the ephemeral public key ($EphemPubKey_U$) to party V in the first message of the key establishment process.
2. Upon receiving party U's ephemeral public key, party V generates a MacTag ($MacTag_V$) and sends it to party U. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData*:

   $$MacData_V = 02 \parallel V \parallel U \parallel Null \parallel EphemPubKey_U \parallel [Text_1]$$

   where *Null* is a null (empty) string.
3. Upon receiving $MacTag_V$, party U verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party U has assurance that party V has derived the same secret *Z* value as party U, and that party V is actively participating in the key establishment process.

8.3.7 C(0,2) Scheme with Bilateral Key Confirmation

Figure 12 depicts the scheme flow for a C(0,2) scheme with bilateral key confirmation. In a C(0,2) scheme, party U (the scheme initiator) and party V (the scheme responder) each have static key pairs. Therefore, either party may assume the role of the provider in a bilateral key confirmation process. However, neither party has an ephemeral key pair; therefore, with this key confirmation method, ephemeral information is provided by each party as a nonce.

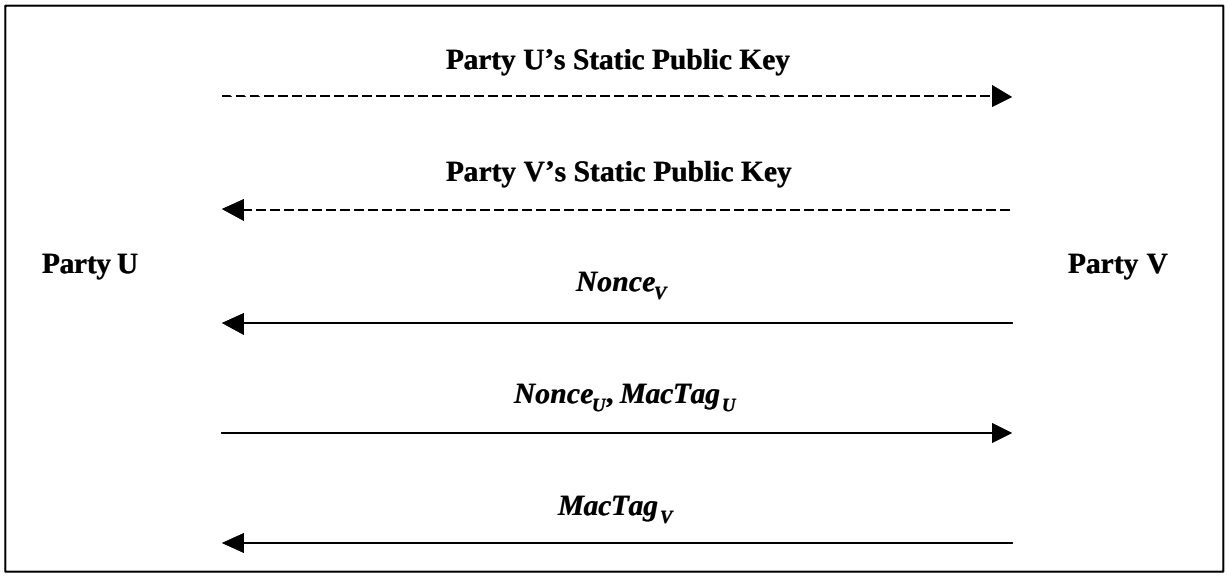

Figure 12: C(0,2) Scheme with Bilateral Key Confirmation

In this method, party U and party V assume the roles of both the provider and recipient in order to obtain bilateral key confirmation. The successful completion of this process provides both parties with assurance that the other party has derived the same secret *Z* value and that each party has actively[5] participated in the key establishment process.

In this scheme, each party has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

The flow proceeds as follows:

1. Party V generates a nonce ($Nonce_V$) and sends it to party U in the first message of the key establishment process.
2. Upon receiving party V's nonce, party U generates a nonce ($Nonce_U$) and a MacTag ($MacTag_U$) and sends them to party V. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData*:

$$MacData_U = 02 \| U \| V \| Nonce_U \| Nonce_V \| [Text_1]$$

3. Upon receiving party U's nonce and the MacTag, party V verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party V

[5] Assurance of active participation is obtained only if the number of unpredictable bits of a nonce is equal to or greater than the desired security level. For example, if 80 bits of security is desired, party V obtains assurance of active participation by Party U only if the number of unpredictable bits of the nonce sent by Party V to Party U is equal to or greater than 80; party U obtains assurance of active participation by Party V only if the number of unpredictable bits of the nonce sent by Party U to Party V is equal to or greater than 80.

has assurance that party U has derived the same secret *Z* value as party V, and that party U is actively participating in the key establishment process.

4. Party V then generates a MacTag ($MacTag_V$) as specified in Sections 5.4.1 and 8.1 and sends the MacTag to party U. The *MacData* for the MacTag is:

$$MacData_V = 03 \| V \| U \| Nonce_V \| Nonce_U \| [Text_2]$$

5. Upon receiving $MacTag_V$, party U verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then U has assurance that party V has derived the same secret *Z* value as party U, and that party V is actively participating in the key establishment process.

8.3.8 C(0,2) Scheme with Unilateral Key Confirmation

Figure 13 depicts the scheme flow for a C(0,2) scheme with unilateral key confirmation. In a C(0,2) scheme, both parties have static key pairs. Therefore, either party may assume the role of the provider in a key confirmation process. However, neither party has an ephemeral key pair; therefore, with this key confirmation method, ephemeral information is provided by the key confirmation recipient as a nonce.

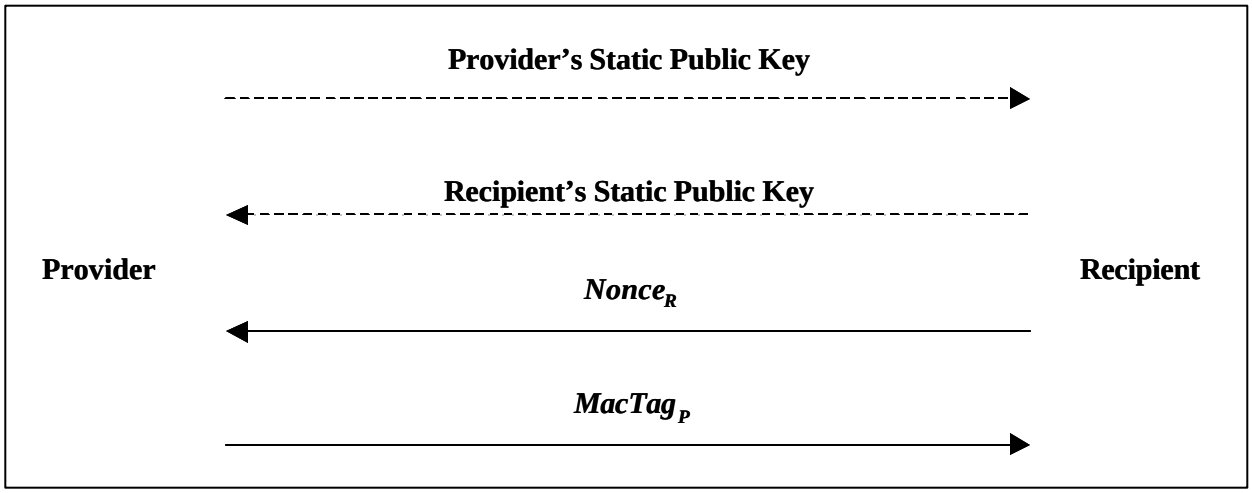

Figure 13: C(0,2) Scheme with Unilateral Key Confirmation

The successful completion of the key confirmation process assures the recipient that the provider has derived the same secret *Z* value as the recipient, and that the provider has actively participated[6] in the key establishment process.

[6] The key confirmation recipient obtains assurance of active participation by the provider only if the number of unpredictable bits of the nonce sent by the recipient to the provider is equal to or greater than the desired security level.

In this scheme, each party has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters. Each party **shall** obtain the other party's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

The flow proceeds as follows:

1. The recipient generates a nonce ($Nonce_R$) and sends it to the provider in the first message of the key establishment process.
2. Upon receiving the nonce, the provider generates a MacTag ($MacTag_P$) and sends it to the recipient. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData*:

   $$MacData_P = 02 \,||\, P \,||\, \mathrm{R} \,||\, Null \,||\, Nonce_R \,||\, [Text_1]$$

   where *Null* is a null (empty) string.
3. Upon receiving $MacTag_P$, the recipient verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then the recipient has assurance that the provider has derived the same secret Z value as the recipient, and that the provider is actively participating in the key establishment process.

8.4 Incorporating Key Confirmation in the DLC-based Key Transport Scheme with Unilateral Key Confirmation

Figure 14 depicts the data scheme flow for a key transport scheme based on Section 7.2 with unilateral key confirmation from the receiver (party V) to the sender (party U). In a key transport scheme, the sender selects the keying material and is assumed to be able to use it correctly; however, the sender may wish to confirm that the receiver was able to correctly unwrap the wrapped keying material. Party V has a static key pair and assumes the role of the key confirmation provider to Party U. Party U has an ephemeral key pair and assumes the role of the key confirmation recipient. If it is appropriate for the key agreement scheme selected, Party U sends a static public key to the receiver along with the ephemeral public key. For key confirmation to be possible, a MAC key **shall** be included in the keying material that is sent by Party U.

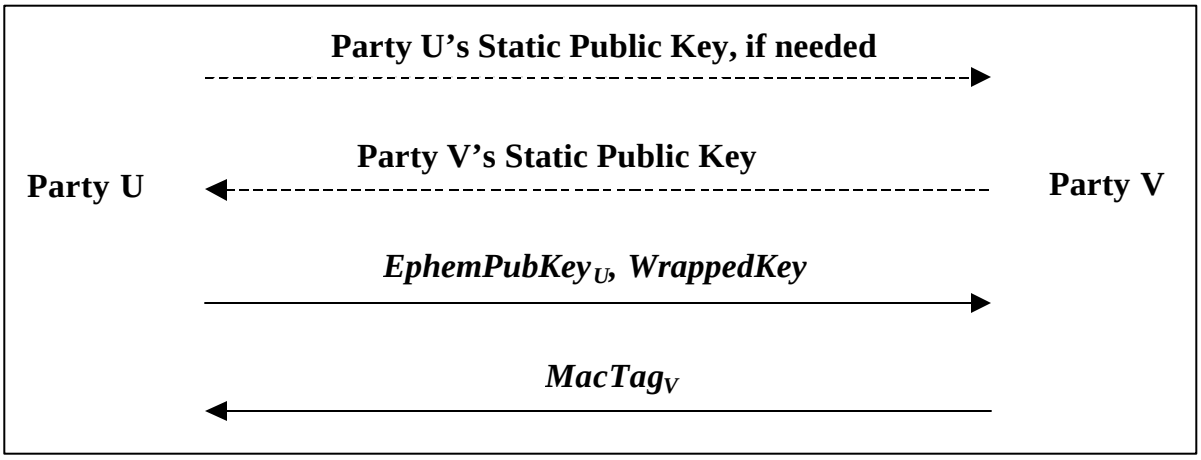

Figure 14: DLC Key Transport Scheme with Unilateral Key Confirmation

This method of key confirmation may be used to provide party U (who is the key confirmation recipient for this method) with assurance that party V (who is the provider) has correctly unwrapped the wrapped keying material sent from party U, and that party V is actively participating in the key establishment process.

In this scheme, party V always has a static key pair that was previously generated as specified in Section 5.2.1; party U **shall** obtain the party V's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA. If the scheme selected is such that party U has a static key, then party U has a static key pair that was previously generated as specified in Section 5.2.1 using the same domain parameters as party V. Party V **shall** obtain party U's static public key in a trusted manner, e.g., from a certificate signed by a trusted CA.

The flow proceeds as follows:

1. Party U follows the procedure in Section 7.2, which includes generating an ephemeral key pair and creating the wrapped keying material. The keying material **shall** include a MAC key. Party U sends $EphemPubKey_U$ and *WrappedKey* to party V in the first message of the key transport process.
2. Upon receiving $EphemPubKey_U$, party V unwraps the *WrappedKey*, recovers the keying material, obtains the MAC key, generates a MacTag ($MacTag_V$) and sends it to party U. The MacTag is generated as specified in Sections 5.4.1 and 8.1 using the following as the *MacData*:

   $$MacData_V = 02 \,||\, V \,||\, U \,||\, Null \,||\, EphemPubKey_U \,||\, [Text_1]$$

   where *Null* is a null (empty) string.
3. Upon receiving *MacTag*, party U verifies the MacTag (see Section 5.4.2). If the received and computed MacTags have the same value, then party U has assurance that party V has correctly unwrapped the keying material that party U sent, and that party V is actively participating in the key establishment process.

9. Key Recovery

For some applications, the keying material used to protect data may need to be recovered (e.g., if the normal reference copy of the keying material is lost or corrupted). In this case, either the keying material or sufficient information to reconstruct the keying material needs to be available (e.g., the keys, domain parameters and other inputs to the scheme used to perform the key establishment process).

Keys used during the key establishment process **shall** be handled in accordance with the following:

1. A static key pair **may** be saved (see the Key Management Guideline [8] for recommended protections); for example, a static public key could be saved in a public key certificate.
2. An ephemeral public key **may** be saved.
3. An ephemeral private key **shall** be destroyed after use and, therefore, is not recoverable.

4. A symmetric key **may** be saved.

Note: This implies that keys derived from schemes where both parties generate ephemeral key pairs (see Section 6.1) cannot be made recoverable by reconstruction of the keying material. For those schemes where only the initiator generates an ephemeral key pair (see Section 6.2), only the responder can recover the keying material by reconstruction.

General guidance on key recovery and the protections required for each type of key is provided in the Key Management Guideline [8].

10. Implementation Validation

Implementations of the schemes in this Recommendation **shall** be tested and validated as conforming to this Recommendation in order to claim compliance with this Recommendation. Information on NIST's cryptographic module testing program is available at http://csrc.nist.gov/cryptval/.

Appendix A: Summary of Differences between this Recommendation and ANSI X9 Standards (Informative)

This list is informational and not meant to be exhaustive, but is intended to summarize important differences between this Recommendation and the indicated ANSI X9 standards. In general, this Recommendation can be seen as being more restrictive than the ANSI X9 standards, but is derived from them. The list of differences is as follows:

1. Random generation and validation of FFC and ECC domain parameters are being extended to (A) support use of the SHA-512 algorithm for domain parameters supporting larger key sizes, (B) support optional use of the Shawe-Taylor algorithm to construct and validate FFC primes and (C) support verifiably random generation of the generator of the subgroup. See the ANSI X9.30-2 DSA revision draft, the ANSI X9.62-2 ECDSA revision draft, and the FIPS 186-3 DSS draft.
2. Some schemes in ANSI X9.42 and X9.63 allow one set of domain parameters to be used with static keys and a different set of domain parameters to be used with ephemeral keys in the same scheme. This Recommendation, however, requires the use of only one set of domain parameters in a scheme; i.e., the same set of domain parameters **shall** be used with the static and ephemeral keys in any given scheme. See Section 5.1 of this Recommendation for more information.
3. For FFC domain parameters: (A) The allowable key sizes for p (field order) are multiples of 1024 bits, rather than multiples of 256 bits as in ANSI X9.42. (B) The size of q (subgroup order) **shall** be a specific length based on the size of p, unlike ANSI X9.42 where the size of q has a minimum length based on the size of p. FFC domain parameters that conform to this Recommendation in this area also conform to ANSI X9.42, although the reverse is not necessarily true. See Section 5.1.1.1 of this Recommendation for more information.
4. For ECC domain parameters: The cofactor **shall** be 65,536 or less, which is more restrictive than X9.63. ECC domain parameters that conform to this Recommendation also conform to ANSI X9.63, although the reverse is not necessarily true. See Section 5.1.1.2 of this Recommendation for more information.
5. Assurances of the arithmetic validity of a public key are required in this Recommendation. Assurance of validity is optional in ANSI X9.42, but required in ANSI X9.63. In both cases, the means of obtaining that assurance is different than in this Recommendation. See Section 5.2.2 of this Recommendation for more information.
6. Methods for a user to receive assurance of possession of the private key associated with a given public key will be specified in this Recommendation. See Section 5.2.3 of this Recommendation for a placeholder.

7. Revised requirements are specifically listed for keys, including requirements specific to static keys and requirements specific to ephemeral keys. See Section 5.2.4 of this Recommendation for details.
8. Regarding the key derivation function (KDF):
    a. The concatenation key derivation function (KDF) of Section 5.3.1 is the default; the ASN.1 KDF of Section 5.3.2 (from X9.42) is optional; it is to be used only when both parties agree on its use. ANSI X9.42 contains both methods, but does not indicate a preference. ANSI X9.63 specifies only the concatenation method in Section 5.6.3 of this Recommendation.
    b. The KDF in this Recommendation **requires** the input of the identifiers of the communicating parties; such information is not required in ANSI X9.42 and X9.63, although these identifiers could be considered a part of the *SharedInfo* field, which is optional and not defined as to contents.
    c. The shared secret **shall** be zeroized before outputting any portion of the *DerivedKeyingMaterial* output; this implies that the entire *DerivedKeyingMaterial* **shall** be computed before outputting any portion of it. The ANSI standards do not indicate when the shared secret needs to be zeroized, deleted or destroyed.

    A KDF that conforms with this Recommendation also conforms to ANSI X9.42, although the reverse is not necessarily true. The use of the default concatenation KDF that conforms with this Recommendation also conforms to ANSI X9.63, although the reverse is not necessarily true. See Section 5.3 of this Recommendation for more details.
9. ANSI X9.42 defines *MacData* as "ANSI X9.42 Testing Message". ANSI X9.63 does not address implementation validation at this level of detail. Note that the implementation test message used for NIST validation is a different text string from the implementation test message for ANSI X9.42; therefore conformance to the method in this Recommendation does not conform with the ANSI X9.42 method, although it does not preclude it. See Section 5.4.3 of this Recommendation for more information.
10. ANSI X9.63 specifies both cofactor and non-cofactor methods. For this Recommendation, ECC cofactor methods **shall** be used. The use of a method that conforms to this Recommendation also conforms to ANSI X9.63, although the reverse is not necessarily true. See Section 6 of this Recommendation for details.
11. FFC and ECC key transport use an Approved key-wrapping algorithm, such as the AES key wrapping algorithm. ANSI X9.63 specifies the ECIES method, which is not allowed in this Recommendation. ANSI X9.42 does not specify a key transport method. Therefore, the use of the method that conforms to this Recommendation does **not** conform to the method in ANSI X9.63. See Section 7.2 for details.
12. There is a comprehensive specification in this Recommendation of approved ways to do Key Confirmation (KC), when KC is desired. See Section 8 of this Recommendation for details. Key confirmation is not discussed in ANSI X9.42, but a few examples of key confirmation are provided in ANSI X9.63.

Appendix B: References (Informative)

[1] FIPS 140-2, Security requirements for Cryptographic Modules, May 25, 2001.

[2] FIPS 180-2, Secure Hash Standard, August 2002.

[3] FIPS 186-3 (Draft), Digital Signature Standard, anticipated in spring 2003.

[4] FIPS 196, Entity Authentication Using Public Key Cryptography, February 1997.

[5] FIPS 197, Advanced Encryption Standard, November 2001.

[6] FIPS 198, The Keyed-Hash Message Authentication Code (HMAC), March 2002.

[7] NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation, December 2001.

[8] NIST SP 800-57 (Draft), Key Management Guideline, January 2003.

[9] ANSI X9.42-2001, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.

[10] ANSI X9.44 (Draft), Public Key Cryptography for the Financial Services Industry: Agreement and Key Transport Using Factoring-Based Cryptography, December 2003.

[11] ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Key Cryptography.

[12] ANSI X9.80-2002, Prime number Generation, Primality Testing and Primality Certificates (Revised).

[13] ANSI X9.82 (Draft), Random Bit Generation, 2003.

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

[15] B. Schneier, Applied Cryptography (Second Edition), John Wiley & Sons, Inc., 1996.

[16] AES Key Wrap Specification, 16 November 2001, currently available at http://csrc.nist.gov/encryption/kms/key-wrap.pdf

[17] NIST Special Publication 800-38B DRAFT Recommendation for Block Cipher Modes of Operation: The RMAC Authentication Mode, November 4, 2002, currently available at http://csrc.nist.gov/publications/drafts/draft800-38B-110402.pdf

[18] ANSI X9.30-2 (Draft) Digital Signature Algorithm (Revised), December 2003.

[19] ANSI X9.62-2 (Draft) Elliptic Curve Digital Signature Algorithm (Revised), December 2002.