NIST Special Publication 800-56A

August 2012

Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography

(Draft Revision)

National Institute of Standards and Technology

Elaine Barker, Lily Chen, Miles Smid and Allen Roginsky

C O M P U T E R S E C U R I T Y

Abstract

This Recommendation specifies key-establishment schemes based on the discrete logarithm problem over finite fields and elliptic curves, including several variations of Diffie-Hellman and MQV key establishment schemes.

KEY WORDS: assurances; Diffie-Hellman; elliptic curve cryptography; finite field cryptography; key agreement; key confirmation; key derivation; key establishment; key management; key recovery; key transport; MQV.

Acknowledgements

The authors gratefully acknowledge and appreciate the contributions by Rich Davis of the National Security Agency and Dustin Moody of the National Institute of Standards and Technology.

Authority

This document has been developed by the National Institute of Standards and Technology (NIST) in furtherance of its statutory responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347.

NIST is responsible for developing standards and guidelines, including minimum requirements, for providing adequate information security for all agency operations and assets, but such standards and guidelines shall not apply to national security systems.

This Recommendation has been prepared for use by federal agencies. It may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright. (Attribution would be appreciated by NIST.)

Nothing in this document should be taken to contradict standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official.

Conformance testing for implementations of this Recommendation will be conducted within the framework of the Cryptographic Module Validation Program (CMVP) and the Cryptographic Algorithm Validation Program(CAVP). The requirements of this Recommendation are indicated by the word "shall." Some of these requirements may be out-of-scope for CMVP or CAVP validation testing, and thus are the responsibility of entities using, implementing, installing or configuring applications that incorporate this Recommendation.

Table of Contents

Figures

Tables

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 specified in NIST Special Publication (SP) 800-67, 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 keying material between the participating entities in advance. Trusted couriers may manually distribute this secret keying material. However, as the number of entities using a system grows, the work involved in the distribution of the secret keying material could grow rapidly. Therefore, it is essential to support the cryptographic algorithms used in modern U.S. Government applications with automated key-establishment schemes.

A key-establishment scheme can be characterized as either a key-agreement scheme or a key transport scheme. The asymmetric-key-based key-establishment schemes in this Recommendation are based on the Diffie-Hellman (DH) and Menezes-Qu-Vanstone (MQV) algorithms. Asymmetric-key-based key-establishment schemes are also specified in SP 800-56B, *Recommendation for Pair-Wise Key-establishment Schemes Using Integer Factorization Cryptography*. The selection of schemes specified in this Recommendation is based on standards for key-establishment schemes developed by the Accredited Standards Committee (ASC) X9, Inc.: ANS X9.42, *Agreement of Symmetric Keys using Discrete Logarithm Cryptography*, and ANS X9.63, *Key Agreement and Key Transport using Elliptic Curve Cryptography*.

2. Scope and Purpose

This Recommendation provides the specifications for key-establishment schemes that are appropriate for use by the U.S. Federal Government and is intended for use in conjunction with NIST Special Publication 800-57, *Recommendation for Key Management* [SP 800-57]. This Recommendation (i.e., SP 800-56A) and the Recommendation for Key Management [SP 800-57] are intended to provide sufficient information for a vendor to implement secure key establishment using asymmetric algorithms in FIPS 140 [FIPS 140] validated modules.

A scheme may be a component of a protocol, which in turn provides additional security properties not provided by the scheme when considered by itself. Note that protocols, per se, are not specified in this Recommendation.

3. Definitions, Symbols and Abbreviations

3.1 Definitions

Approved	FIPS-**approved** or NIST-Recommended. An algorithm or technique that is either 1) specified in a FIPS or NIST Recommendation, or 2) adopted in a FIPS or NIST Recommendation and specified either (a) in an appendix to the FIPS or NIST Recommendation, or (b) in a document referenced by the FIPS or NIST Recommendation.
Assumption	This term is used to indicate the conditions that are required to be true when an **approved** key-establishment scheme is executed in accordance with this Recommendation.
Assurance of private-key possession	Confidence that an entity possesses a private key corresponding to a public key.
Assurance of validity	Confidence that either a key or a set of domain parameters is arithmetically correct.
Binding	Assurance of the integrity of an asserted relationship between items of information that is provided by cryptographic means. Also see Trusted association.
Bit length	The length in bits of a bit string.
Bit string	An ordered sequence of 0's and 1's.
Byte	A bit string consisting of eight bits. A byte is represented by a hexadecimal string of length two. The right-most hexadecimal character represents the rightmost four bits of the byte, and the left-most hexadecimal character of the byte represents the left-most four bits of the byte. For example, 9d represents the bit string 10011101.
Byte string	An ordered sequence of bytes.
Certification Authority (CA)	The entity in a Public-Key Infrastructure (PKI) that is responsible for issuing public key certificates and exacting compliance to a PKI policy.
Cofactor	The order of the elliptic curve group divided by the (prime) order of the generator point (i.e. the base point) specified in the domain parameters.
Destroy	In this Recommendation, to destroy is an action applied to a key or a piece of secret data. After a key or a piece of secret data is destroyed, no information about its value can be recovered.

Domain parameters	The parameters used with a cryptographic algorithm that are common to a domain of users.
Entity	An individual (person), organization, device, or process. "Party" is a synonym.
Ephemeral key pair	A key pair, consisting of a public key (i.e., an ephemeral public key) and a private key (i.e., an ephemeral private key) that is intended for a very short period of use. The key pair is ordinarily used in exactly one transaction of a cryptographic scheme; an exception to this is when the ephemeral key pair is used in multiple transactions for a key-transport broadcast. Contrast with a static key pair.
Hash function	A function that maps a bit string of arbitrary length to a fixed-length bit string. **Approved** hash functions are expected to satisfy the following properties: 1. One-way: It is computationally infeasible to find any input that maps to any pre-specified output, and 2. Collision resistant: It is computationally infeasible to find any two distinct inputs that map to the same output. **Approved** hash functions are specified in [FIPS 180].
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 (for example, a nickname or a string consisting of an IP address).
Key agreement	A (pair-wise) key-establishment procedure in which the resultant secret keying material is a function of information contributed by both participants, so that neither party can predetermine the value of the secret keying material independently from the contributions of the other party. Contrast with key transport.
Key-agreement transaction	An execution of a key-agreement scheme.
Key confirmation	A procedure to provide assurance to one party (the key confirmation recipient) that another party (the key confirmation provider) actually possesses the correct secret keying material and/or shared secret.
Key confirmation provider	The party that provides assurance to the other party (the recipient) that the two parties have indeed established a shared secret or shared keying material.

Key derivation method	The process by which keying material is derived from a shared secret and other information.
Key establishment	The procedure that results in keying material that is shared among different parties.
Key-establishment key pair	A private/public key pair used in a key-establishment scheme. It can be a static key pair or an ephemeral key pair.
Key-establishment transaction	An execution of a key-establishment scheme. It can be either a key-agreement transaction or a key-transport transaction.
Key transport	In this Recommendation, a (pair-wise) key-establishment procedure whereby one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver). Contrast with key agreement.
Key-transport transaction	An execution of a key-transport scheme.
Key wrapping	In this Recommendation, key wrapping is a method of protecting keying material using a symmetric-key-based authenticated encryption method, such as a block cipher key-wrapping mode specified in [SP 800-38F] that provides both confidentiality and integrity protection.
Key-wrapping key	In this Recommendation, a key-wrapping key is a symmetric key established through a key-agreement transaction and used with a key-wrapping algorithm to protect the keying material to be transported.
Keying material	Data that is represented as a binary string such that any non-overlapping segments of the string with the required lengths can be used as symmetric cryptographic keys. In this Recommendation, keying material is derived from a shared secret established during an execution of a key-establishment scheme or generated by the sender in a key transport scheme. As used in this Recommendation, secret keying material may include keys, secret initialization vectors, and other secret parameters.
MAC tag	Data obtained from the output of a MAC algorithm that can be used by an entity to verify the integrity and the origination of the information used as input.

Message Authentication Code (MAC) algorithm	A MAC algorithm is a family of cryptographic functions – parameterized by a symmetric key – that can be used to provide data origin authentication, as well as data integrity, by producing a MAC tag on arbitrary data (the message). In this Recommendation, an **approved** MAC algorithm is used for key confirmation and may also be employed in certain key derivation methods.
Nonce	A time-varying value that has at most a negligible chance of repeating. For example, the nonce may be a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.
Owner	For a static public key, static private key and/ or the static key pair containing those components, the owner is the entity that is authorized to use the static private key corresponding to the static public key, whether that entity generated the static key pair itself or a trusted party generated the key pair for the entity. For an ephemeral key pair, ephemeral private key or ephemeral public key, the owner is the entity that generated the ephemeral key pair and uses the ephemeral private key associated with the public key of that key pair.
Party	See entity.
Public-key certificate	A data structure that contains an entity's identifier(s), the entity's public key (including an indication of the associated set of domain parameters) and possibly other information, along with a signature on that data set that is generated by a trusted party, i.e. a certificate authority, thereby binding the public key to the included identifier(s).
Receiver	The party that receives secret keying material via a key-transport transaction. Contrast with sender.
Recipient	A party that (1) receives a public key; or (2) obtains assurance from an assurance provider (e.g., assurance of the validity of a candidate public key or assurance of possession of the private key corresponding to a public key); or (3) receives key confirmation from a key confirmation provider.
Scheme	A (cryptographic) scheme consists of an unambiguous specification of a set of transformations that are capable of providing a (cryptographic) service when properly implemented and maintained. A scheme is a higher-level construct than a primitive and a lower-level construct than a protocol.
Security strength (Also "Bits of security")	A number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system.

Security properties	The security features (e.g., entity authentication or key confirmation) that a cryptographic scheme may, or may not, provide.
Sender	The party that sends secret keying material to the receiver in a key-transport transaction.
Shall	This term is used to indicate a requirement of a Federal Information Processing Standard (FIPS) or a requirement that needs to be fulfilled to claim conformance to this Recommendation. Note that **shall** may be coupled with **not** to become **shall not**.
Shared secret	A secret value that has been computed using a key-agreement scheme and is used as input to a key derivation method.
Should	This term is used to indicate an important recommendation. Ignoring the recommendation could result in undesirable results. Note that **should** may be coupled with **not** to become **should not**.
Static key pair	A key pair, consisting of a private key (i.e., a static private key) and a public key (i.e., a static public key) that is intended for use for a relatively long period of time and is typically intended for use in multiple key establishment transactions. Contrast with an ephemeral key pair.
Symmetric-key algorithm	A cryptographic algorithm that uses a single secret key for different operations, such as for encryption and decryption.
Trusted association	Assurance of the integrity of an asserted relationship between items of information that may be provided by cryptographic or non-cryptographic (e.g., physical) means. Also see Binding.
Trusted party	A trusted party is a party that is trusted by an entity to faithfully perform certain services for that entity. An entity could be a trusted party for itself.
Trusted third party	A third party, such as a CA, that is trusted by its clients to perform certain services. (By contrast, in a key establishment transaction, the participants, parties U and V, are considered to be the first and second parties.)

3.2 Symbols and Abbreviations

General:

AES	Advanced Encryption Standard (as specified in [FIPS 197]).
ASC	The American National Standards Institute (ANSI) Accredited Standards Committee.

ANS	American National Standard.
ASN.1	Abstract Syntax Notation One.
C(*i*e)	Notation for a category of key-establishment schemes, in which *i* ephemeral key pairs are used, where $i \in \{0, 1, 2\}$.
C(*i*e,*j*s)	Notation for a subcategory of key-establishment schemes, in which *i* ephemeral key pairs and *j* static key pairs are used. In this Recommendation, schemes in the subcategories C(0e, 2s), C(1e, 2s), C(1e, 1s), C(2e, 0s), and C(2e, 2s) are defined.
CA	Certification Authority.
CDH	The cofactor ECC Diffie-Hellman key-agreement primitive.
DH	The (non-cofactor) FFC Diffie-Hellman key-agreement primitive.
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 operations in an elliptic curve group.
FF	Finite Field.
FFC	Finite Field Cryptography, the public-key cryptographic methods using operations in a multiplicative group of a finite field.
H	An auxiliary function used in certain key derivation methods. H is either an **approved** hash function, *hash*, or an HMAC-*hash* based on an **approved** hash function, *hash*, with a salt value used as the HMAC key.
HMAC-*hash*	Keyed-hash Message Authentication Code (as specified in [FIPS 198]) with an **approved** hash function *hash*.
ID	The bit string denoting the identifier associated with an entity.
KC	Key Confirmation.
KDF	Key Derivation Function.
MAC	Message Authentication Code.

MAC(*MacKey*, *MacData*)	A MAC algorithm with *MacKey* as the key, and *MacData* as the data.
MacTag	A MAC tag.
MQV	The Menezes-Qu-Vanstone key-agreement primitive.
Null	The empty bit string
SHA	Secure Hash Algorithm (as specified in [FIPS 180]).
$T_{bitLen}(X)$	A truncation function that outputs the most significant (i.e., leftmost) *bitLen* bits of the input bit string, X, when the bit length of X is greater than *bitLen*; otherwise, the function outputs X. For example, $T_2(1011) = 10$, $T_3(1011) = 101$, and $T_4(1011) = 1011$.
TTP	Trusted Third Party.
U, V	Represent the two parties in a (pair-wise) key establishment scheme.
$\{X\}$	Indicates that the inclusion of X is optional.
$X \| Y$	Concatenation of two bit strings X and Y.
$[a, b]$	The set of integers x, such that $a \leq x \leq b$.
$\lceil x \rceil$	The ceiling of x; the smallest integer $\geq x$. For example, $\lceil 5 \rceil = 5, \lceil 5.3 \rceil = 6$.
Z	A shared secret (represented as a byte string) that is used to derive secret keying material using a key derivation method.
Z_e	A component of the shared secret (represented as a byte string) that is computed using ephemeral keys in a Diffie-Hellman primitive.
Z_s	A component of the shared secret (represented as a byte string) that is computed using static keys in a Diffie-Hellman primitive.

The following notations are used for FFC and ECC in this Recommendation. Note that the notation sometimes differs between the two scheme types, due to the differing notations used in the two standards on which this Recommendation is based (i.e., ANS X9.42 and ANS X9.63).

FFC:

GF(p)	The finite field with p elements, where p is an (odd) prime number. The elements of $GF(p)$ can be represented by the set of integers $\{0, 1, \ldots, p\text{-}1\}$. The addition and multiplication operations for $GF(p)$ can be realized by performing the corresponding integer operations and reducing the results modulo p.
*GF(p)**	The multiplicative group of non-zero field elements in $GF(p)$.
g	An FFC domain parameter; the selected generator of the multiplicative subgroup of prime order q in $GF(p)^*$.
$k \bmod p$	The modular reduction of the (arbitrary) integer k by the (positive) integer p (the modulus). For the purposes of this Recommendation, $j = k \bmod p$ is the unique integer satisfying the following two conditions: $0 \leq j < p$ and $k - j$ is a multiple of p.
p	An FFC domain parameter; an odd prime number that determines the size of the finite field $GF(p)$.
pgenCounter	An FFC domain parameter; a value that may be output during domain parameter generation to provide assurance at a later time that the resulting domain parameters were generated using a canonical process.
q	An FFC domain parameter; q is the (odd) prime number equal to the order of the multiplicative subgroup of *GF(p)** generated by the FFC domain parameter g. Note that q is a divisor of $p - 1$.
r_U, r_V	The ephemeral private keys of party U and party V, respectively. These are integers in the range $[2, q\text{-}2]$.
t_U, t_V	The ephemeral public keys of party U and party V, respectively. These are integers in the range $[2, p\text{-}1]$, representing elements in the finite field $GF(p)$.
SEED	An FFC domain parameter; an initialization value that is used during domain parameter generation that can also be used to provide assurance at a later time that the resulting domain parameters were generated using a canonical process.
x_U, x_V	The static private keys of party U and party V, respectively. These are integers in the range $[2, q\text{-}2]$.
y_U, y_V	The static public keys of party U and party V, respectively. These are integers in the range $[2, p\text{-}1]$, representing elements in the finite field $GF(p)$.

ECC:

a, b	ECC domain parameters; two elements in the finite field *GF(q)* that define the equation of an elliptic curve, $y^2 = x^3 + ax + b$ when q is a prime or $y^2 + xy = x^3 + ax^2 + b$, when $q = 2^m$ for an integer m.
avf(Q)	The associate value of the elliptic curve point Q.
$d_{e,U}$, $d_{e,V}$	The ephemeral private keys of party U and party V, respectively. These are integers in the range [2, n-2].
$d_{s,U}$, $d_{s,V}$	The static private keys of party U and party V, respectively. These are integers in the range [2, n-2].
FR	Field Representation indicator (an ECC domain parameter); an indication of the basis used for representing field elements. FR is NULL if the field has odd prime order or if a Gaussian normal basis is used. If a polynomial basis representation is used for a field of order 2^m, then FR indicates the reduction polynomial (a trinomial or a pentanomial).
G	An ECC domain parameter, which is a distinguished (affine) point in an elliptic curve group that generates a subgroup of prime order n.
GF(q)	The finite field with q elements, where either q is an odd prime p or q is equal to 2^m, for some prime integer m. The elements of *GF(q)* are represented by the set of integers $\{0, 1, \ldots, p-1\}$ in the case that q is an odd prime p, or as bit strings of length m bits in the case that $q = 2^m$.
h	An ECC domain parameter; the cofactor, a positive integer that is equal to the order of the elliptic curve group, divided by the order of the cyclic subgroup generated by the distinguished point G. That is, nh is the order of the elliptic curve, where n is the order of the cyclic subgroup generated by the distinguished point G.
n	An ECC domain parameter; a prime that is the order of the cyclic subgroup generated by the distinguished point G.
Ø	The "point at infinity"; a special element of an elliptic curve group that serves as the (additive) identity.
q	An ECC domain parameter; the field size. Either q is an odd prime p or q is equal to 2^m, for some prime integer m.
$Q_{e,U}$, $Q_{e,V}$	The ephemeral public keys of party U and party V, respectively. These are points on the elliptic curve defined by the domain parameters.

$Q_{s,U}$, $Q_{s,V}$	The static public keys of party U and party V, respectively. These are points on the elliptic curve defined by the domain parameters.
SEED	An optional ECC domain parameter; an initialization value that is used during domain parameter generation that can also be used to provide assurance at a later time that the resulting domain parameters were generated using a canonical process.
x_P, y_P	Elements of the finite field *GF(q)*, representing the x and y coordinates, respectively, of a point *P*.

4. Overview of Key-Establishment Schemes

Secret cryptographic keying material may be electronically established between parties by using a key-establishment scheme, that is, by using either a key-agreement scheme or a key-transport scheme.

During a pair-wise key-agreement scheme, the secret keying material to be established is not sent directly from one entity to another. Instead, the two parties exchange information from which they each compute a shared secret that is used (along with other exchanged/known data) to derive the secret keying material. The method used to combine the information made available to both parties provides assurance that neither party can control the output of the key-agreement process.

The key-agreement schemes described in this Recommendation employ public-key techniques utilizing Discrete Logarithm Cryptography (DLC). The security of these DLC-based key-agreement schemes depends upon the intractability of the discrete logarithm problem in certain settings.

In this Recommendation, the **approved** key-agreement schemes are described in terms of the roles played by parties "U" and "V." These are specific labels that are used to distinguish between the two participants engaged in key agreement – irrespective of the actual labels that may be used by a protocol employing a particular **approved** key-agreement scheme.

To be in conformance with this Recommendation, a protocol employing any of the approved pair-wise key-agreement schemes shall unambiguously assign the roles of U and V to the participants by clearly defining which participant performs the actions ascribed by this Recommendation to party U, and which performs the actions ascribed herein to party V.

During key transport, one party selects the secret keying material to be transported. The secret keying material is then wrapped using a shared key-wrapping key and an approved key-wrapping algorithm (in particular, it is encrypted with integrity protection) and sent to the other party. The party that selects, wraps, and sends the secret keying material is called the "sender," and the other party is called the "receiver." The key-transport techniques described in this Recommendation combine a DLC key-agreement scheme with a key-wrapping technique. First, an approved key-agreement scheme is used to establish a key-wrapping key that is shared between party U and party V. Then, party U (now acting as the key-transport sender) wraps the

keying material that will be transported, using an approved key-wrapping algorithm; party V (acting as the key-transport receiver) later uses the same key-wrapping key to unwrap the transported keying material. (See Section 7 for details, including restrictions on the key-agreement schemes that are approved for such key-transport applications.)

This Recommendation specifies a number of processes that are associated with key establishment (including processes for generating domain parameters and for deriving secret keying material from a shared secret). Some of these processes are used to provide assurance (for example, assurance of the arithmetic validity of a public key or assurance of possession of a private key associated with a public key). The party that provides the assurance is called the "provider" (of the assurance), and the party that obtains the assurance is called the "recipient" (of the assurance). For any of the specified processes, equivalent processes 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.

Sections 4.1, 4.2, and 4.3 describe the various steps that may be performed to establish secret keying material.

4.1 Key Establishment Preparations

The owner of a private/public key pair is the entity that is authorized to use the private key of that key pair. The precise steps required may depend upon the key establishment scheme and the type of key pair (static or ephemeral).

The first step is to obtain appropriate domain parameters that are generated as specified in Section 5.5.1; either the owner itself generates the domain parameters, or the owner obtains domain parameters that another entity (e.g., a trusted third party) has generated. Having obtained the domain parameters, the owner obtains assurance of the validity of those domain parameters; **approved** methods for obtaining this assurance are provided in Section 5.5.2.

If the owner will be using a key establishment scheme that requires that the owner have a static key pair, the owner obtains this key pair. Either the owner or a trusted third party generates the key pair as specified in Section 5.6.1. If the key pair is generated by a trusted third party, then the key pair **shall** be transported to the owner in a protected manner. If the key establishment scheme requires an ephemeral key pair, the owner generates it (as close to the time of use as possible) as specified in Section 5.6.1. Before using a static or ephemeral key pair in a key-establishment transaction, its owner is required to confirm its validity by obtaining the assurances specified in Section 5.6.2.1.

An identifier is used to label the entity that owns a static key pair used in a key establishment transaction; an identifier may also be used to label the owner of an ephemeral key pair. This label may uniquely distinguish the owner from all other entities, in which case it could rightfully be considered an identity. However, the label may be something less specific – an organization, nickname, etc. – hence, the term identifier is used in this Recommendation, rather than the term identity. For example, an identifier could be "Vegetable.gardener123", rather than an identifier that names a particular person. A key pair's owner (or an agent trusted to act on the owner's behalf) is responsible for ensuring that the identifier associated with its static public key is appropriate for the applications in which it will be used.

For each static key pair, this Recommendation assumes that there is a trusted association between the intended owner's identifier(s) and the intended owner's static public key. The association may be provided using cryptographic mechanisms or by physical means. The use of cryptographic mechanisms may require the use of a binding authority (i.e., a trusted authority) that binds the information in a manner that can be verified by others; an example of such a trusted authority is a registration authority working with a CA who creates a certificate containing both the static public key and the identifier. The binding authority **shall** verify the owner's intent to associate a specific identifier chosen for the owner and the public key; the means for accomplishing this is beyond the scope of this Recommendation. The binding authority **shall** also obtain assurance of the validity of the domain parameters associated with the owner's key pair, the arithmetic validity of the owner's static public key, and the owner's possession of the static private key corresponding to that static public key (see Section 5.5.2, Section 5.6.2.2.1 [method 1], and Section 5.6.2.2.3, respectively.)

As an alternative to reliance upon a binding authority, trusted associations between identifiers and static public keys may be established by the direct exchange of this information between entities, using a mutually trusted method (e.g., a trusted courier or a face-to-face exchange). In this case, each entity receiving an identifier and the associated static public key **shall** be responsible for obtaining the same assurances that would have been obtained on their behalf by a binding authority (see the previous paragraph). Entities **shall** also be responsible for maintaining (by cryptographic or other means) the trusted associations between any identifiers and static public keys received through such exchanges.

If an entity engaged in a key-establishment transaction owns a static key pair that is employed during the transaction, then the identifier used to label that party **shall** be one that has a trusted association with the static public key of that key pair. If an entity engaged in a key-establishment transaction contributes only an ephemeral public key during the transaction, but an identifier is still desired/required for that party, then a non-null identifier **shall** be selected/assigned in accordance with the requirements of the protocol relying upon the transaction.

Figure 1 depicts the steps that may be required of an owner in order to obtain its key pair(s) in preparation for key establishment.

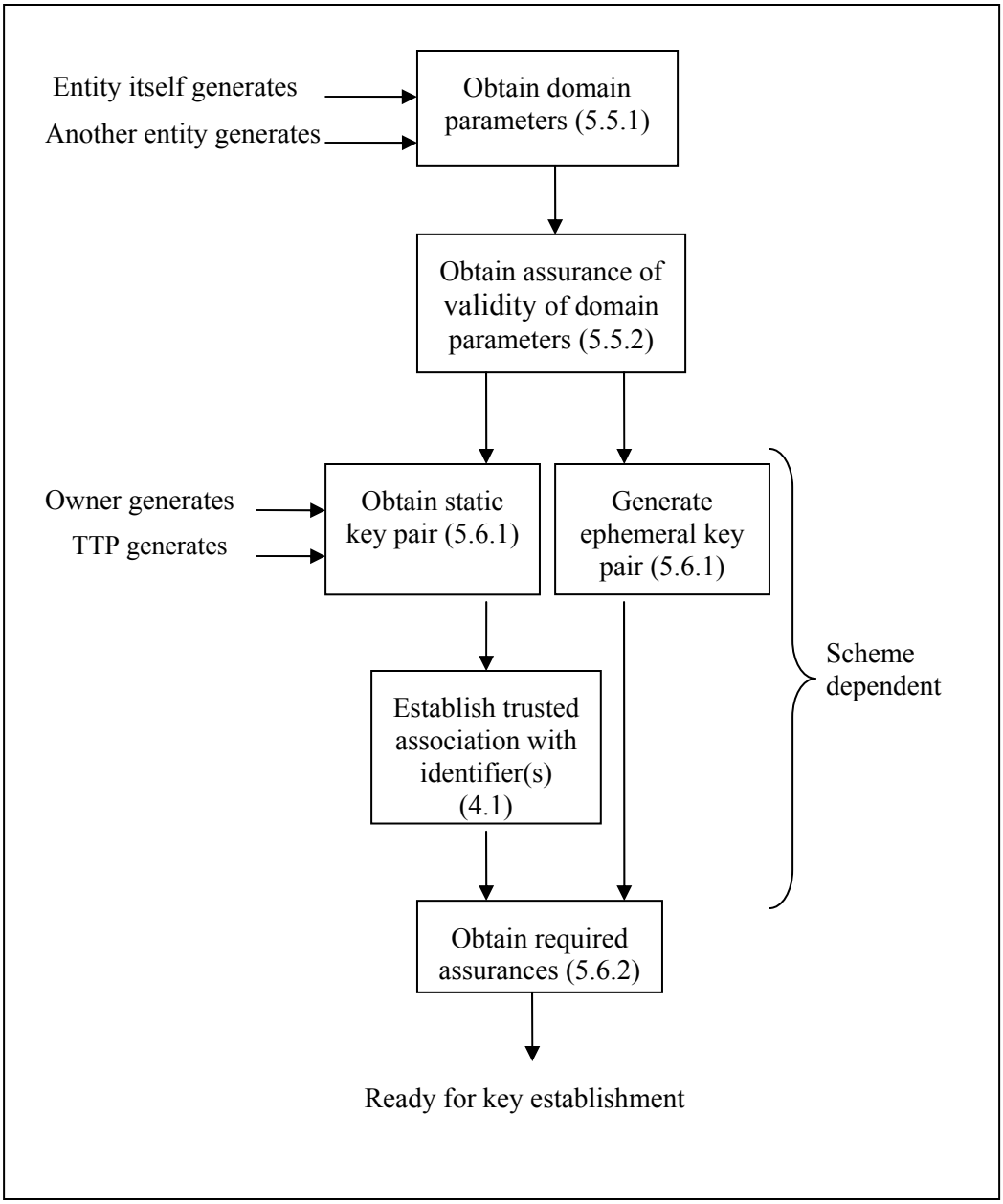

Figure 1: Owner Key Establishment Preparations

4.2 Key Agreement Process

Some discrepancies in the order of the steps may occur, however, depending upon the communication protocol in which the key-agreement process is performed. Depending on the key-agreement scheme and the available keys, the party whose actions are described could be either of the two participants in the key-agreement scheme (i.e., either party U or party V). Note that some of the actions shown in Figure 2 may be absent from certain schemes. For example, key confirmation as defined in Section 5.9 is optional, which is indicated with dashed frames and

dashed arrows. The specifications of this Recommendation indicate when a particular action is required.

If required by the key agreement scheme, a party that requires the other entity's static public key acquires that key (as well as the associated identifier) and obtains assurance of its validity. **Approved** methods for obtaining assurance of the validity of the other entity's static public key are provided in Section 5.6.2.2.1. Assurance that the other entity is in possession of the corresponding static private key must also be obtained prior to using the derived keying material for purposes beyond those of the key agreement transaction itself. See Section 5.6.2.2.3 for **approved** methods for obtaining this assurance. (Note: the restriction above does not, for example, prohibit the use of derived keying material for key confirmation performed *during* the key agreement transaction.).

If a party receives an ephemeral public key from the other entity for use in the key agreement transaction, that party must obtain assurance of its validity. **Approved** methods for obtaining assurance of the validity of the other entity's ephemeral public key are provided in Section 5.6.2.2.2

If required by the key agreement scheme, a party generates an ephemeral key pair (in accordance with Section 5.6.1) and provides the ephemeral public key of that key pair to the other entity; the ephemeral private key is not provided to the other entity.

If required or desired for use in the key agreement transaction, a party generates a nonce (as specified in Section 5.4) and provides it to the other entity.

Depending upon the circumstances, additional public information (e.g., a party's static public key, an identifier, etc.) may be provided to or obtained from the other entity.

If either of the participants in the key agreement transaction requires evidence that the other participant has computed the same shared secret and/or derived the same secret keying material, (unilateral or bilateral) key confirmation may be performed as specified in Section 5.9.

Figure 2 depicts the steps that may be required of an entity when establishing secret keying material with another entity by using one of the key-agreement schemes described in this Recommendation.

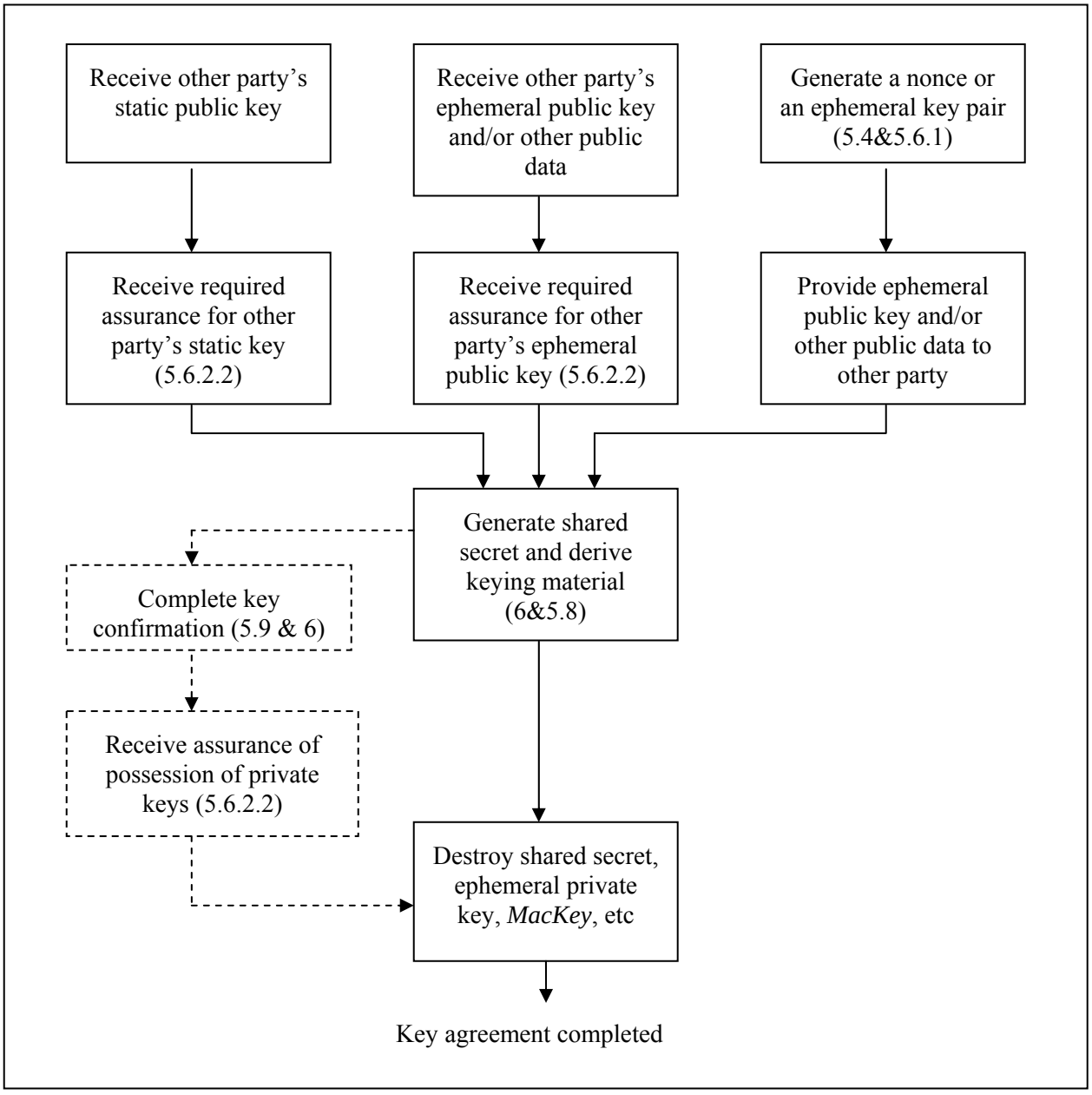

Figure 2: Key Agreement Process

4.3 DLC-based Key Transport Process

The process begins by establishing a key-wrapping key using an appropriate key-agreement scheme (see Sections 6 and 7), with the intended key-transport sender acting as party U, and the intended key-transport receiver acting as party V. Key confirmation may optionally be performed at the end of the key agreement process to provide assurance that both parties possess the same key-wrapping key. Party U then selects secret keying material to be transported, wraps the keying material using the key-wrapping key and sends the wrapped keying material to party

V. After receiving and unwrapping the transported keying material, party V may optionally perform key confirmation to provide assurance to party U that the transported keying material has been received and correctly unwrapped. Figure 3 depicts the steps that are performed when transporting secret keying material from one entity to another using a key transport scheme.

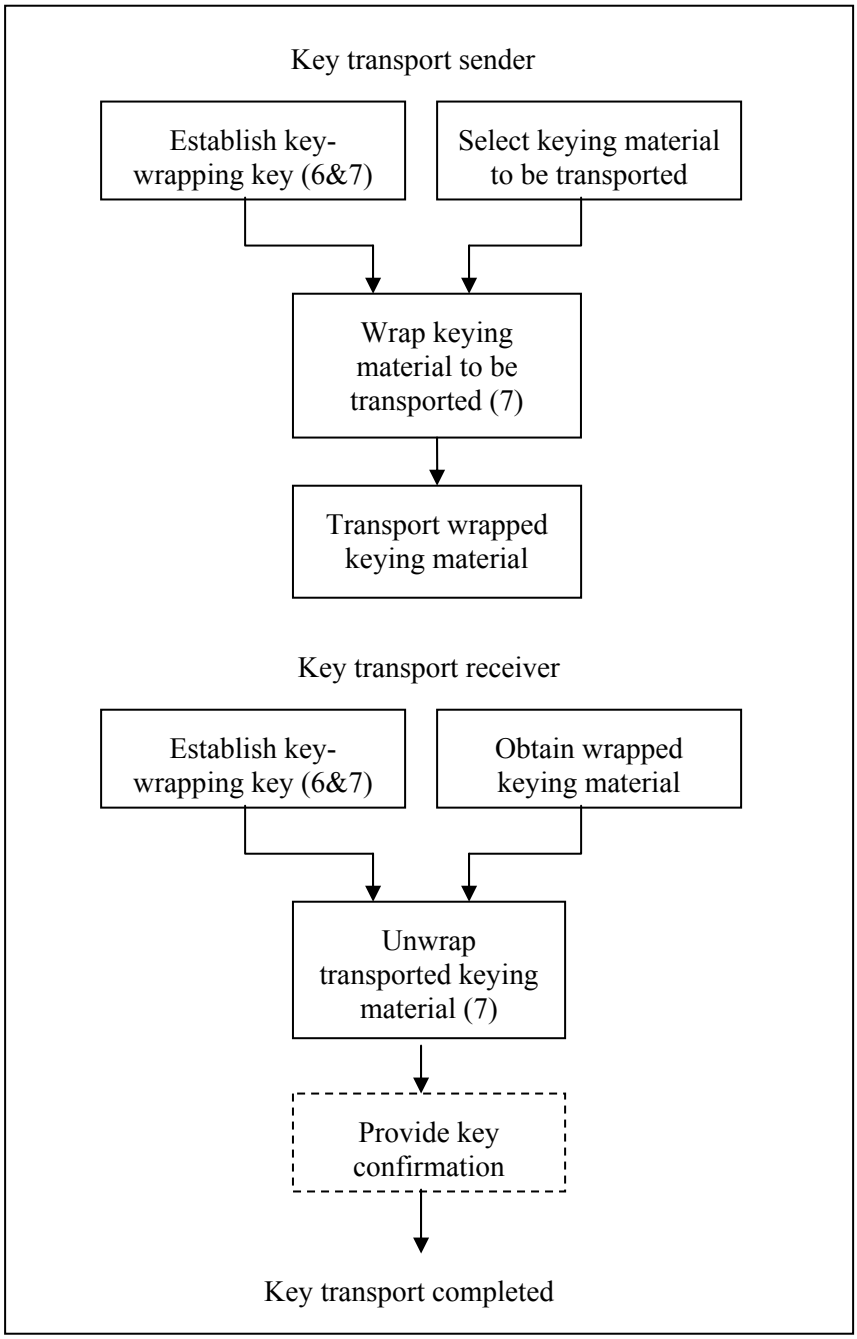

Figure 3: Key Transport Process

5. Cryptographic Elements

This section describes the basic computations that are performed and the assurances that need to be obtained when performing DLC-based key establishment. The schemes described in Section 6 are based upon the correct implementation of these computations and assurances.

5.1 Cryptographic Hash Functions

In this Recommendation, cryptographic hash functions may be used in key derivation and in MAC tag computation during key confirmation. An **approved** hash function **shall** be used when a hash function is required. [FIPS 180] specifies **approved** hash functions.

5.2 Message Authentication Code (MAC) Algorithm

A Message Authentication Code (MAC) algorithm defines a family of MAC functions that is parameterized by a symmetric key. The input to a MAC function includes a symmetric key, called *MacKey* and a binary data string called *MacData*. That is, a MAC function is represented as MAC(*MacKey*, *MacData*). In this Recommendation, a MAC function is used in key confirmation and may be used for key derivation.

Approved MAC algorithms are specified in [FIPS 198] (i.e., HMAC) and [SP 800-38B] (i.e., CMAC). HMAC requires the use of an **approved** hash function; CMAC requires the use of an **approved** block cipher algorithm.

When used for key derivation, HMAC **shall** be selected as the function H in accordance with Table 6 and Table 7 of Section 5.8. The selection of CMAC is defined in [800-56C].

When used for key confirmation, an entity is required to compute a MAC tag on received or derived data using a MAC function determined by a *MacKey* that is derived from a shared secret. The MAC tag is sent to the other entity participating in the key-establishment scheme in order to provide assurance that the shared secret or derived keying material was correctly computed. MAC tag computation and verification are defined in Sections 5.2.1 and 5.2.2.

5.2.1 MAC Tag Computation for Key Confirmation

The computation of the MAC tag is represented as follows:

$$MacTag = T_{MacLen}[\text{MAC}(MacKey, MacData)].$$

The MAC tag computation **shall** be performed using an a**pproved** MAC algorithm (See [FIPS 198] and [SP 800-38B]) to obtain *MacOutputLen* bits, the full output length of the MAC algorithm. The *MacOutputLen* bits may be followed by applying a truncation function T_{MacLen} to obtain *MacLen* bits, where *MacLen* is required to be less than or equal to *MacOutputLen*. In the above equation, MAC represents an a**pproved** MAC algorithm; *MacKey* represents a symmetric key obtained from the *DerivedKeyingMaterial* (see Section 5.8 and Section 5.9.1); *MacLen* represents the length of *MacTag;* and *MacData* represents the data on which the MAC tag is computed. The minimum for *MacLen* and the minimum length of *MacKey* are specified in Table 8 and Table 9 of Section 5.9.3.

5.2.2 MAC Tag Verification for Key Confirmation

To check a received MAC tag, *MacTag* (e.g., received during key confirmation), a new MAC tag, *MacTag′* is computed using the values of *MacKey*, *MacLen*, and *MacData* possessed by the recipient (as specified in Section 5.2.1). *MacTag′* is compared with the received *MacTag*. If their values are equal, then it may be inferred that the same *MacKey*, *MacLen*, and *MacData* values were used in the two MAC tag computations.

5.3 Random Number Generation

Whenever this Recommendation requires the use of a randomly generated value (for example, for obtaining keys or nonces), the values **shall** be generated using an **approved** random bit generator (RBG), such as those specified in [SP800-90A] providing an appropriate security strength.

5.4 Nonces

A nonce is a time-varying value that has (at most) a negligible chance of repeating (where the meaning of "negligible" may be application specific). In certain schemes specified in this Recommendation, a party may be required to provide a (public) nonce that is used for key-agreement and/or key-confirmation purposes. This circumstance arises when a scheme does not require that party to provide an ephemeral public key to the other party as part of the key-establishment process.

This Recommendation requires the use of a nonce (supplied by Party U) in the C(0e, 2s) key-agreement schemes specified in Section 6.3. A nonce (supplied by party V) is also required by the C(1e, 2s) and C(0e, 2s) schemes when party V obtains key confirmation from party U in conformance with this Recommendation (see Section 6.2.1.5 and Section 6.3.3, respectively).

A nonce may be composed of one (or more) of the following components (other components may also be appropriate):

1. A random value that is generated anew for each nonce, using an **approved** random bit generator. A nonce containing a component of this type is called a *random nonce*.
2. A timestamp of sufficient resolution (detail) so that it is different each time it is used.
3. A monotonically increasing sequence number, or
4. A combination of a timestamp and a monotonically increasing sequence number, such that the sequence number is reset when and only when the timestamp changes. (For example, a timestamp may show the date but not the time of day, so a sequence number is appended that will not repeat during a particular day.)

The specified use of a nonce in key-derivation and/or key-confirmation computations does not provide the same benefits as the use of an ephemeral key pair in a key-agreement scheme. (For example, party U's contribution of a public nonce during the execution of a C(0e, 2s) scheme does not protect the secrecy of derived keying material against a future compromise of party U's static private key, but the use of an ephemeral key pair by party U during the execution of a C(1e, 2s) scheme can provide such protection.) Still, the contribution of an appropriately formed

nonce can support some of the security goals (e.g., assurance of the "freshness" of derived keying material) that might otherwise be supported by the contribution of an ephemeral public key generated (and used) in conformance with this Recommendation.

Whenever it is required for key-agreement and/or key-confirmation purposes as specified in this Recommendation, a nonce **should** be a random nonce. The security strength supported by the instantiation of this random bit generator and the bit length of the random bit string **shall** be at least one-half of the minimum bit length required for the subgroup order, as determined by the applicable parameter-size set. However, the bit length of the random bit string **should** be (at least) equal to the bit length of the subgroup order included among the domain parameters that are actually employed during key agreement (i.e., $\lceil \log_2 q \rceil$ in the FFC case, or $\lceil \log_2 n \rceil$ in the ECC case). The minimum bit lengths required for the subgroup order by various parameter-size sets are specified in Table 1 of Section 5.5.1.1 (for the FFC case) and in Table 2 of Section 5.5.1.2 (for the ECC case). For details concerning the security strength supported by an instantiation of a random bit generator, see [SP 800-90A].

As part of the proper implementation of this Recommendation, system users and/or agents trusted to act on their behalf **should** determine that the components selected for inclusion in any required nonces meet their security requirements. The application tasked with performing key establishment on behalf of a party **should** determine whether or not to proceed with a key-establishment transaction based upon the perceived adequacy of the method(s) used to form the required nonces. Such knowledge may be explicitly provided to the application in some manner, or may be implicitly provided by the operation of the application itself.

5.5 Domain Parameters

Discrete Logarithm Cryptography (DLC), which includes 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 candidate set of domain parameters is said to be valid when it conforms to all the requirements specified in this Recommendation. Both parties executing a key-establishment scheme **shall** have assurance of domain parameter validity prior to using them (e.g., to generate key pairs). 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 pairs and with multiple key-establishment schemes.

For this Recommendation, only one set of domain parameters **shall** be used during any key establishment transaction. That is, when a key establishment scheme uses both a static key pair and an ephemeral key pair, they **shall** be generated using the same set of domain parameters.

5.5.1 Domain Parameter Generation

5.5.1.1 FFC Domain Parameter Generation

If p is a prime number, then $GF(p)$ denotes the finite field with p elements, which can be represented by the set of integers $\{0, 1, \ldots, p-1\}$. The addition and multiplication operations for $GF(p)$ can be realized by performing the corresponding integer operations and reducing the

results modulo p. The multiplicative group of non-zero field elements is denoted by *GF(p)**. In this Recommendation, an FFC key-establishment scheme requires the use of public keys that are restricted to a (unique) cyclic subgroup of *GF(p)** with prime order q (where q divides $p - 1$). If g is a generator of this cyclic subgroup, then its elements can be represented as $\{1, g \bmod p, g^2 \bmod p, \ldots, g^{q-1} \bmod p\}$, and $1 = g^q \bmod p$.

Domain parameters for an FFC scheme are of the form (p, q, g{, *SEED, pgenCounter*}), where p is the (odd) prime field size, q is an (odd) prime divisor of $p - 1$, g is a generator of the cyclic subgroup of *GF(p)** of order q, and *SEED* and *pgenCounter* are optional parameters used in the canonical process of generating and validating p, q, and possibly g (depending on the method of generation). FFC Domain parameters **shall** be generated using a method specified in [FIPS 186], based on a parameter-size set selected from Table 1 (i.e., parameter set FA, FB or FC).

Table 1: FFC Parameter-Size Sets

FFC Parameter-Size Set Name	FA	FB	FC
Bit length of field size p (i.e., $\lceil \log_2 p \rceil$)	1024	2048	2048[1]
Bit length of subgroup order q (i.e., $\lceil \log_2 q \rceil$)	160	224	256

As shown in Table 1, there are three parameter-size sets (named FA, FB, and FC). For U.S. government applications, one or more of the parameter-size sets provided in Table 1 **shall** be used for the FFC schemes, based on security strength requirements. See the comparable security table in the Recommendation for Key Management [SP 800-57] to assess the comparable security of any particular parameter-size set. [SP 800-57] provides guidance on selecting an appropriate security strength and an appropriate FFC parameter set[2]. If an FFC parameter-size set cannot provide the required security strength, then Elliptic Curve Cryptography **should** be used (see Section 5.5.1.2).

For this Recommendation, the size of p (the public key size) is either 1024 or 2048 bits. The size of q is a specific bit length, depending on the FFC parameter-size set selected. The leftmost (i.e., most significant) bit of p and q **shall** be a "1" (e.g., for a 1024-bit prime p, where bit 1 is the most significant bit and bit 1024 is the least significant, bit 1 shall be a 1).

5.5.1.2 ECC Domain Parameter Generation

For ECC, let $GF(q)$ denote the finite field with q elements, where either q is an odd prime p, or q is equal to 2^m, for some prime integer m. An elliptic curve defined over $GF(q)$ is determined by

[1] Parameter-size set FC is included with the same bit length for the field size as set FB to allow finite field applications with a 2048-bit field order to have the option of increasing the private key size to 256 bits without having to increase the field order (a more substantial change). FC is not intended to provide more security than FB; i.e., parameter sets FB and FC provide keys with the same security strength.

[2] In [SP 800-57], l is used to denote the bit length of the field size, and N denotes the bit length of the subgroup order.

either an equation of the form $y^2 = x^3+ax+b$ (when $q = p$) or by an equation of the form $y^2+xy = x^3+ax^2+b$ (when $q = 2^m$), where a and b are (appropriately chosen) elements of $GF(q)$. For the purposes of this Recommendation, an affine point P on the corresponding elliptic curve is one that can be represented as an ordered pair (x_P, y_P), whose coordinates are elements of $GF(q)$ that satisfy the given equation. Together with an appropriate binary operation "+" (elliptic-curve addition) and the introduction of a special point $Ø$ (the "point at infinity") that serves as the (additive) identity, the set of (affine and infinite) elliptic curve points forms a group.

As specified in this Recommendation, an ECC key-establishment scheme requires the use of public keys that are affine elliptic curve points chosen from a specific cyclic subgroup with prime order n. Suppose that the point G is a generator for this cyclic subgroup. If, for each positive integer d, dG denotes $G + G +\ldots+ G$ (the elliptic-curve sum of d copies of G), then the elements of the cyclic subgroup can be represented as $\{Ø, G, 2G,\ldots, (n-1)G\}$. Note that $nG = Ø$. The full elliptic curve group (including $Ø$ and all of the affine points on the curve) has order nh, where the integer h is called a *cofactor* of the cyclic subgroup generated by G.

Domain parameters for an ECC scheme have the form $(q, FR, a, b\{, SEED\}, G, n, h)$. The parameter q is the field size. As noted above, q may be an odd prime p, or q may be equal to 2^m, for some prime integer m. The field representation parameter FR is an indication of the method used to represent elements of the finite field $GF(q)$. FR is null if q is equal to an odd prime p. In this case, the elements of the finite field are represented by the integers 0 through $p - 1$. FR is also null if $q = 2^m$, and $GF(2^m)$ uses a Gaussian normal basis representation (of the type specified in [ANS X9.62]). If $q = 2^m$, and $GF(2^m)$ uses a polynomial basis representation, then FR is the reduction polynomial – either a trinomial or a pentanomial. The parameters a and b are elements of $GF(q)$ that define the equation of an elliptic curve. $G = (x_G, y_G)$ is an affine point on the elliptic curve determined by a and b that is used to generate a cyclic subgroup of prime order n. The parameter h is the cofactor of the cyclic subgroup generated by G. The bit string $SEED$ is an optional parameter used in the canonical process of generating and validating a, b, and possibly G (depending on the method of generation).

The ECC domain parameters **shall** either be generated as specified in [ANS X9.62] or selected from the recommended elliptic curve domain parameters specified in [FIPS 186]. (Note: ANS X9.62, rather than ANS X9.63, specifies the most current methods of generating ECC domain parameters at the time of writing this revision of the Recommendation.)

Table 2: ECC Parameter-Size Sets

ECC Parameter-Size Set Name	EA	EB	EC	ED	EE
Bit length of ECC subgroup order n (i.e., $\lceil \log_2 n \rceil$)	160-223	224-255	256-383	384-511	512+
Maximum bit length of ECC cofactor h	10	14	16	24	32

As shown in Table 2, there are five parameter-size sets (named EA, EB, EC, ED and EE) for ECC. The five different cofactor maximums each ensure that the subgroup of order n is unique

and that cofactor multiplication is reasonably efficient. For U.S. government applications, one or more of the parameter-size sets provided in Table 2 **shall** be used for the ECC schemes, based on security strength requirements. See the comparable security strength table in the Recommendation for Key Management [SP 800-57] to assess the comparable security of any particular parameter-size set. The Recommendation for Key Management [SP 800-57] provides guidance on selecting the appropriate security strength and an appropriate ECC key size.

5.5.2 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 as specified in Section 5.5.1.
2. The party performs an explicit domain parameter validation as specified in:
   a. [FIPS 186] for FFC, based on a parameter-size set selected from Table 1.
   b. [ANS X9.62] for ECC, based on a parameter-size set selected from Table 2.
3. The party has received assurance from a trusted third party (for example, a CA or NIST[3]) that the set of domain parameters was valid at the time that they were generated by reason of either method 1 or 2 above.

As part of the proper implementation of this Recommendation, system users and/or agents trusted to act on their behalf **should** determine which of the methods above meet their security requirements. The application tasked with performing key establishment on behalf of a party **should** determine whether or not to proceed with a key establishment transaction, based upon the perceived adequacy of the method(s) used to obtain assurance of domain parameter validity. Such knowledge may be explicitly provided to the application in some manner, or may be implicitly provided by the operation of the application itself.

5.5.3 Domain Parameter Management

The set of domain parameters used **shall** be protected against modification or substitution until the set is deactivated (if and when it is no longer needed). Each private/public key pair **shall** be correctly associated with its specific set of domain parameters.

5.6 Key-Establishment Key Pairs

This section specifies requirements for the generation of key pairs to be used in key-establishment transactions, provides methods for obtaining assurances that valid key pairs are used during key establishment, and specifies key management requirements for the static and ephemeral key pairs used in key establishment.

[3] NIST-recommended curves can be found in [FIPS 186].

5.6.1 Key-Pair Generation

These generation methods assume the use of valid domain parameters (see Section 5.5). Prior to performing key-pair generation with the selected domain parameters, the party generating the key pair **shall** obtain assurance of domain parameter validity in accordance with Section 5.5.2.

5.6.1.1 FFC Key-Pair Generation

For the FFC schemes, each static and ephemeral key pair **shall** be generated using an **approved** method (see Appendix B of [FIPS 186]) and the selected valid domain parameters (i.e., *p, q, g*{, *SEED, pgenCounter*}). Given a set of valid domain parameters, each valid private key is a randomly selected integer in the range [2, q–2]. Each valid static public key y is related to the corresponding (valid) static private key x by the following formula: $y = g^x \bmod p$. Similarly, each valid ephemeral public key t is related to the corresponding (valid) ephemeral private key r by the following formula: $t = g^r \bmod p$.

5.6.1.2 ECC Key-Pair Generation

For the ECC schemes, each static and ephemeral private key d and public key Q **shall** be generated using an **approved** method (see Appendix B of [FIPS 186]) and the selected domain parameters (i.e., *q, FR, a, b*{, *SEED*}, *G, n, h*). Given valid domain parameters, each valid private key d is an integer that is randomly selected in the range [2, n–2]. Whether static or ephemeral, each valid public key Q is related to the corresponding (valid) private key d by the following formula: $Q = (x_Q, y_Q) = dG$.

5.6.2 Required Assurances

To explain the assurance requirements associated with key-establishment key pairs, some terminology needs to be introduced. The owner of a static key pair is defined as the entity that is authorized to use the private key that corresponds to the public key; this is independent of whether or not the owner generated the key pair. The recipient of a static public key is defined as the entity that is participating in a key establishment transaction with the owner and obtains the key before or during the current transaction. The owner of an ephemeral public key is the entity that generated the key as part of a key-establishment transaction. The recipient of an ephemeral public key is the entity that receives that public key during a key-establishment transaction with its owner.

Secure key establishment depends upon the use of valid key-establishment keys. Prior to obtaining the assurances described in this section, the owner of a key pair and the recipient of the public key of that key pair **shall** obtain assurance of the validity of the associated domain parameters (see Section 5.5.2).

The security of key-agreement schemes also depends on limiting knowledge of the private keys to those who have been authorized to use them (i.e., their respective owners) and to the trusted third party that may have generated them. In addition to preventing unauthorized entities from gaining access to private keys, it is also important that owners have access to their private keys.

Note that as time passes, an owner may lose possession of the correct value of the private key component of their key pair, either by choice or due to an error; for this reason, current assurance of possession of a static private key can be of value for some applications, and renewing

assurance of possession may be necessary. See Section 5.6.2.2.3.2 for techniques that the recipient of a static public key can use to directly obtain more current assurance of the owner's possession of the corresponding private key.

Prior to or during a key-establishment transaction, the participants in the transaction (i.e., parties U and V**) shall** obtain the appropriate assurances about the key pairs used during that transaction. The types of assurance that may be sought by one or both of the parties (U and/or V) concerning the components of a key pair (i.e., the private key and public key) are discussed in Sections 5.6.2.1 and 5.6.2.2. The methods that will be specified to provide/obtain these assurances presuppose the validity of the domain parameters associated with the key pair (see Section 5.5).

The following sections include tables that summarize the types of assurance that are required by the parties to a key-establishment transaction. Table 4 in Section 5.6.2.1 summarizes assurances that a key-pair owner may want to renew periodically. The shaded table entries indicate a type of key pair (static or ephemeral) and a type of assurance that might be sought for such a key pair. The unshaded table entries indicate who can perform the actions necessary to obtain the assurance.

5.6.2.1 Assurances Required by the Key Pair Owner

Prior to the use of a static or ephemeral key pair in a key-establishment transaction, the key-pair owner **shall** confirm the validity of the key pair by obtaining the following assurances:

- Assurance of correct generation – assurance that the key pair was generated as specified in Section 5.6.1 (see Section 5.6.2.1.1 for the methods for obtaining this assurance).
- Assurance of private-key validity – assurance that the private key is an integer in the correct range, as determined by the domain parameters (see Section 5.6.2.1.2 for the methods for obtaining this assurance).
- Assurance of public-key validity – assurance that the public key has the correct representation for a non-identity element of the correct cryptographic subgroup, as uniquely determined by the domain parameters (see Section 5.6.2.1.3 for the methods for obtaining this assurance).
- Assurance of pair-wise consistency – assurance that the private key and public key have the correct mathematical relationship to each other (see Section 5.6.2.1.4 for the methods for obtaining this assurance).

Table 3 indicates the assurances to be obtained by the owner of a key pair for both static and ephemeral keys, identifies who can perform the actions necessary for the owner to obtain each assurance, and indicates the sections of this document where further information is provided.

Table 3: Initial Assurances Required by the Key-Pair Owner

Key-Pair Type	Types of Assurance			
	Correct Generation	**Private-Key Validation**	**Public-Key Validation**	**Pair-wise Consistency**
Static	Owner[a] or TTP[b]	Owner[c]	Owner[d] or TTP[e]	Owner[f]
Ephemeral	Owner[a]	Owner[c]	Owner[d]	Owner[f]

a See Section 5.6.2.1.1, method a.
b See Section 5.6.2.1.1, method b
c See Section 5.6.2.1.2
d See Section 5.6.2.1.3, method a and b.
e See Section 5.6.2.1.3, method c.
f See Section 5.6.2.1.4.

A static key pair owner may optionally renew certain assurances regarding its key pair at any time. Table 4 indicates which of the assurances obtained by the owner of a static key pair can be renewed and indicates the sections of this document where further information is provided. Note that for ephemeral key pairs, only initial assurances are required; renewed assurance for ephemeral key pairs is not applicable, since ephemeral key pairs are short-lived. Also, note that the assurance of the correct generation of a static key pair is not renewable since, after the fact, it is not feasible to verify that its private component was randomly selected.

Table 4: Optional Renewal of Assurances by the Key-Pair Owner

Key-Pair Type	Types of Assurance			
	Correct Generation	**Private-Key Validation**	**Public-Key Validation**	**Pair-wise Consistency**
Static	Infeasible	Owner[a]	Owner[b]	Owner[c]

a. See Section 5.6.2.1.2.
b. See Section 5.6.2.1.3.
c. See Section 5.6.2.1.4.

Note that the methods used to obtain the required assurances are not necessarily independent. For example, the key-pair owner may employ a key-generation routine that is consistent with the criteria of Section 5.6.1 and also incorporates the actions required to provide (initial) assurance of the validity and consistency of the private and public components of the resulting key pair.

As part of the proper implementation of this Recommendation, system users and/or agents trusted to act on their behalf **should** determine which of the methods above meet their security requirements. The application tasked with performing key establishment on behalf of a party

should determine whether or not to proceed with a key-establishment transaction, based upon the perceived adequacy of the method(s) used to obtain the above assurances.

5.6.2.1.1 Owner Assurance of Correct Generation

Prior to the use of a key pair in a key-establishment transaction, the owner of a static or ephemeral key-establishment key pair **shall** obtain an initial assurance that the key pair has been correctly formed (in a manner that is consistent with the criteria of Section 5.6.1) using one of the following methods:

a. The owner generates the key pair as specified in Section 5.6.1 (this applies to both static and ephemeral key pairs), or

b. A trusted third party (trusted by the owner and any recipient of the public key) generates a static key pair as specified in Section 5.6.1 and provides it to the owner. Note that, in this case, the TTP needs to be trusted by both the owner and any public-key recipient to generate the key pair as specified in Section 5.6.1 and not to use the owner's private key to masquerade as the owner. This method is not appropriate for ephemeral key pairs, since the owner generates ephemeral keys.

5.6.2.1.2 Owner Assurance of Private-Key Validity

Prior to the use of a key pair in a key-establishment transaction, the owner of a static or ephemeral key-establishment key pair **shall** obtain an initial assurance that the private key is an integer in the correct range ([2, q–2] for FFC or [2, n–2] for ECC), as determined by the domain parameters (see Sections 5.5.1.1 and 5.5.1.2) using one of the following methods:

a. The owner generates the key pair as specified in Section 5.6.1 (this applies to both static and ephemeral key pairs), or

b. After receiving the static key pair from a trusted third party (trusted by the owner), the owner performs a separate check to determine that the private key is in the correct range; note that only static key pairs may be generated by a TTP.

To renew this assurance for a static key pair, the owner **shall** perform a separate check to determine that the private key is in the correct range as determined by the domain parameters.

5.6.2.1.3 Owner Assurance of Public-Key Validity

Prior to a key-establishment transaction, the owner of a key pair **shall** obtain an initial assurance that the static or ephemeral public key has the (unique) correct representation for a non-identity element of the correct cryptographic subgroup, as determined by the domain parameters, using one of the following methods:

a. The owner generates the key pair as specified in Section 5.6.1 and performs a full public-key validation or an equivalent procedure as part of its generation process (see Sections 5.6.2.3.1 and 5.6.2.3.2), or

b. The owner performs a full public-key validation as a separate process from the key-pair generation process (see Sections 5.6.2.3.1 and 5.6.2.3.2); either the owner or a TTP could have generated the key pair; or

c. A trusted third party (trusted by the owner) performs a full public-key validation (see Sections 5.6.2.3.1 and 5.6.2.3.2) and provides the validation result to the owner. This TTP could, for example, be a binding authority (see Section 4.1) and/or a TTP that generated the key pair (see method b in Section 5.6.2.1.1). In the case of TTP generation, the TTP may employ a key-generation routine that performs a full public-key validation or an equivalent procedure as part of its key-pair generation process, or the full public-key validation may be performed as a separate process from the key-pair generation process following the generation of the key pair.

To renew this assurance for a static public key (if desired), the owner **shall** perform a successful full public-key validation (see Sections 5.6.2.3.1 and 5.6.2.3.2). Note that renewed assurance of validity for an ephemeral public key is not applicable, since ephemeral key pairs are short-lived.

5.6.2.1.4 Owner Assurance of Pair-wise Consistency

Prior to a key-establishment transaction, the owner of a key pair **shall** obtain an initial assurance that the private key and public key have the correct mathematical relationship to each other by using one of the following methods:

a. The owner generates the key pair as specified in Section 5.6.1 (this applies to both static and ephemeral key pairs), or

b. Subsequent to key generation as specified in Section 5.6.1 by the owner or a trusted third party, the owner performs one of the following alternative consistency tests. If a consistency test fails, the tested key pair **shall not** be used.

   Alternative 1:

   The public key **shall** be recomputed from the private key and the domain parameters in order to obtain assurance that the private and public keys are consistent.

   FFC schemes: Check that $g^x \bmod p = y$, where x is the private key, y is the associated public key of the key pair, and both the generator g and prime modulus p are domain parameters associated with the key pair.

   ECC schemes: Check that $dG = Q$, where d is the private key Q is the associated public key of the key pair, and the generator G is a domain parameter associated with the key pair.

   The test fails if the check specified above fails.

   Alternative 2:

   Using the static key pair to be tested for pair-wise consistency and the domain parameters associated with that key pair, separately perform the computations that would be required of both party U and party V in an **approved** DLC key-agreement scheme to generate derived keying material. The static key pair to be tested **shall** be used as party V's key pair. Any other key pairs required to generate the shared secret **shall** be

a. pre-determined (generated as specified in Section 5.6.1 using the same domain parameters[4] as those used for the static key pair to be tested for consistency);

b. pair-wise consistent; and

c. used only for pair-wise consistency testing.

The following are the main procedures for the pair-wise consistency test:

1. Select a DLC key-agreement scheme from Section 6 with which the static key pair will be used to perform the pair-wise consistency test.

2. Use the static key pair to be tested and any pre-determined key pairs required for the scheme with the appropriate DLC primitive for that scheme to generate the shared secrets Z_U and Z_V for party U and party V, respectively.

   FCC schemes: Use the DLC key-agreement primitive in either Section 5.7.1.1 or 5.7.2.1, as appropriate for the scheme to be used for the pair-wise-consistency test.

   ECC Schemes: Use the DLC primitive in either Section 5.7.1.2 or 5.7.2.3, as appropriate for the scheme to be used for the pair-wise-consistency test.

   Depending on the scheme selected, the DH schemes specified in Section 6 may require one or two pre-determined key pairs for party U. If the key-agreement scheme requires party V to have only one key pair, then the static key pair to be tested for pair-wise consistency **shall** be used as that key pair. If the scheme requires party V to have two key pairs (i.e., an ephemeral key pair and a static key pair), then the static key pair to be tested **shall** be used as the static key pair, and the other key pair **shall** be a pre-determined key pair distinct from the pre-determined key pair(s) used for party U.

   For the schemes using MQV primitives specified in Section 5.7.2, MQV1 and one-pass form **shall** be used for FFC and ECC primitives, respectively. In both cases, party U requires two pre-determined key pairs. The static key pair to be tested will be used as both of party V's contributions to the primitive. Note that the MQV1 and One-Pass forms of the primitives differ from the MQV2 and Full MQV form of the primitive only in the number of key pairs required by party V; for the MQV1 and One-Pass primitives, party V only uses the key pair to be tested, whereas for MQV2 and Full MQV form, an additional pre-determined key pair is required for party V.

3. Derive keying material K_U using an **approved** key-derivation function (KDF) as specified in Section 5.8 for party U using Z_U; derive keying material K_V using the same KDF for party V using Z_V. The length of K_U and K_V **shall** be at least the length of the KDF output block.

The test fails if $K_U \neq K_V$.

[4] If an implementation allows the use of multiple sets of domain parameters, each set of domain parameters will require a static key pair for pair-wise consistency testing.

To renew assurance of pair-wise consistency for a static key pair (if desired), one of the above alternative consistency tests in method b **shall** be employed by the owner. Note that renewed assurance for ephemeral key pairs is not applicable, since ephemeral key pairs are short-lived.

5.6.2.1.5 Owner Assurance of Possession of the Private Key

Prior to a key-establishment transaction, the owner of a key pair **shall** obtain an initial assurance of possession of the private key using one of the following methods:

a. The owner generates the key pair as specified in Section 5.6.1 (this applies to both static and ephemeral key pairs), or

b. When a trusted third party (trusted by the owner) generates a static key pair and provides it to the owner, the owner performs one of the consistency tests in Section 5.6.2.1.4; if a pair-wise consistency test fails, the tested key pair **shall not** be used.

To renew this assurance for a static private key (if desired), one of the alternative pair-wise consistency tests in method b of Section 5.6.2.1.4 **shall** be employed by the owner. Note that renewed assurance of the possession of an ephemeral private key is not applicable, since ephemeral key pairs are short-lived.

5.6.2.2 Assurances Required by a Public Key Recipient

In order to successfully employ any of the schemes specified in this Recommendation, each participant in a key-establishment transaction must receive at least one public key owned by the other participant. The public key(s) may be received during the transaction (which is usually the case for an ephemeral public key) or prior to the transaction (as is sometimes the case for a static public key). Regardless of the timing, a transaction participant is said to be acting as a "public-key recipient" when it receives the other participant's public key(s). Note that besides the participants (i.e.party U and party V), a binding authority (e.g., a CA) may be a public key recipient (e.g., when obtaining assurance of possession).

Prior to or during a key-establishment transaction, the recipient of a public key **shall** obtain the following indicated assurances:

- Assurance of public-key validity – assurance that the public key of the other party (i.e., the claimed owner of the public key) has the (unique) correct representation for a non-identity element of the correct cryptographic subgroup, as determined by the domain parameters (see Sections 5.6.2.2.1 and 5.6.2.2.2). This assurance is required for both static and ephemeral public keys.

- Assurance of private-key possession – assurance that the claimed owner of a public key-establishment key (i.e., the other party) actually has the (correct) private key associated with that public key. This assurance is required for static private keys (see Section 5.6.2.2.3). Assurance of private-key possession for ephemeral keys is optional; however, methods for obtaining this assurance are discussed in Section 5.6.2.2.4.

Table 5 summarizes the assurances required by a public-key recipient for both the static and ephemeral public keys of the other party, identifying the party that may perform the actions

necessary for the recipient to obtain the assurance and indicating the sections in this document where further information is provided.

Table 5: Assurances Required by a Public-Key Recipient

Key-Pair Type	Type of Assurance	
	Public-Key Validation	**Private-Key Possession**
Static	Recipient[a] or TTP[b]	Recipient[d] or TTP[e]
Ephemeral	Recipient[c]	Not Required[f]

a See Section 5.6.2.2.1, method 1.
b See Section 5.6.2.2.1, method 2.
c See Section 5.6.2.2.2
d. See Section 5.6.2.2.3.2
e. See Section 5.6.2.2.3.1
f However, see Section 5.6.2.2.4

The recipient of a static public key may optionally renew its assurance that the owner of that public key (i.e., the other party) still possesses the static private key associated with the public key. Note that renewed assurance of possession of ephemeral private keys by the other party is not applicable, since ephemeral key pairs are short-lived.

As part of the proper implementation of this Recommendation, system users and/or agents trusted to act on their behalf **should** determine which of the methods above meet their security requirements. The application tasked with performing key establishment on behalf of the recipient **should** determine whether or not to proceed with a key-establishment transaction, based upon the perceived adequacy of the method(s) used to obtain the assurance above.

5.6.2.2.1 Recipient Assurance of Static Public-Key Validity

The recipient of another party's static public key **shall** obtain assurance of the validity of that public key in one or more of the following ways:

1. The recipient performs a successful full public-key validation of the received public key (see Sections 5.6.2.3.1 and 5.6.2.3.2).

2. The recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful full public-key validation of the received public key (see Sections 5.6.2.3.1 and 5.6.2.3.2). This TTP could, for example, be a binding authority, such as a CA (see Section 4.1), and/or the TTP that generated the key pair (in accordance with method b of Section 5.6.2.1.1) and provided it to the owner (i.e., the other party). In the case of TTP generation, the TTP may employ a key-generation routine that performs a full public-key validation or an equivalent procedure as part of its key-pair generation process, or the full public-key validation may be performed as a separate process from the generation of the key pair for the other party.

5.6.2.2.2 Recipient Assurance of Ephemeral Public-Key Validity

The recipient of another party's ephemeral public key **shall** obtain assurance of the validity of that public key using one or more of the following methods:

1. The recipient or a third party trusted by the recipient performs a successful full public-key validation on the received public key (see Sections 5.6.2.3.1 and 5.6.2.3.2).
2. If using an ECC method (only), the recipient or a third party trusted by the recipient performs a successful partial public-key validation on the received public key (see Section 5.6.2.3.3).

5.6.2.2.3 Recipient Assurance of the Owner's Possession of a Static Private Key

The recipient of another party's static public key **shall** obtain an initial assurance that the other party (i.e., the claimed owner of the public key) actually possesses the associated private key, either prior to or concurrently with performing a key-agreement transaction with that other party. Assurance of the validity of the corresponding public key **shall** be obtained prior to obtaining this assurance.

As part of the proper implementation of this Recommendation, system users and/or agents trusted to act on their behalf **should** determine which of the methods for obtaining assurance of possession meet their security requirements. The application tasked with performing key establishment on behalf of a party **should** determine whether or not to proceed with a key-establishment transaction, based upon the perceived adequacy of the method(s) used. Such knowledge may be explicitly provided to the application in some manner, or may be implicitly provided by the operation of the application itself.

If a binding authority is used to bind the key pair owner's identifier to his static public key: At the time of binding an owner's identifier to his static public key, the binding authority (i.e., a trusted third party, such as a CA) **shall** obtain assurance that the owner is in possession of the correct static private key. This assurance **shall** either be obtained using one of the methods specified in Section 5.6.2.2.3.2 (e.g., with the binding authority acting as the public-key recipient) or by using an **approved** alternative (see SP 800-57, Part 1, Sections 5.2 and 8.1.5.1.1.2).

Recipients not acting in the role of a binding authority **shall** obtain this assurance – either through a trusted third party (see Section 5.6.2.2.3.1) or directly from the owner (i.e., the other party) (see Section 5.6.2.2.3.2) – before using the derived keying material for purposes beyond those required during the key-agreement transaction itself. If the recipient chooses to obtain this assurance directly from the other party (i.e., the claimed owner of that public key), then to comply with this Recommendation, the recipient **shall** use one of the methods specified in Section 5.6.2.2.3.2.

5.6.2.2.3.1 Recipient Obtains Assurance from a Trusted Third Party

The recipient of a static public key may receive assurance that its owner (i.e., the other party in the key-agreement transaction) is in possession of the correct static private key from a trusted third party (trusted by the recipient), either before or during a key-agreement transaction that makes use of that static public key. The methods used by a third party trusted by the recipient to

obtain that assurance are beyond the scope of this Recommendation (see however, the discussion in Section 5.6.2.2.3 above and Section 8.1.5.1.1.2 of SP 800-57 [7]).

5.6.2.2.3.2 Recipient Obtains Assurance Directly from the Claimed Owner (i.e., the Other Party)

When two parties engage in a key-agreement transaction, there is (at least) an implicit claim of ownership made whenever a static public key is provided on behalf of a particular party. That party is considered to be a *claimed* owner of the corresponding static key pair – as opposed to being a *true* owner – until adequate assurance can be provided that the party is actually the one authorized to use the static private key. The claimed owner can provide such assurance by demonstrating its knowledge of that private key.

If all of the following conditions are met during a key-agreement transaction that incorporates key confirmation as specified in this Recommendation, then in the course of establishing keying material, the recipient of a static public key may be able to directly obtain (initial or renewed) assurance of the claimed owner's (i.e., the other party's) current possession of the corresponding static private key:

1) The recipient of the static public key contributes an ephemeral public key to the key-agreement process, one that is intended to be arithmetically combined with the claimed owner's (i.e., the other party's) static private key in computations performed by the claimed owner. (If an appropriate key-agreement scheme is employed, the claimed owner will be challenged to demonstrate current knowledge of his static private key by successfully performing those computations during the transaction.)
2) The recipient of the static public key is also a key-confirmation recipient, with the claimed owner (i.e., other party) serving as the key-confirmation provider. (By successfully providing key confirmation, the claimed owner can demonstrate that he is the true owner of the received static public key and that he currently knows the corresponding static private key.)

There are a number of key-agreement schemes specified in this Recommendation that can be used while satisfying both of the conditions above. In order to claim conformance with this Recommendation, the key-agreement transaction during which the recipient of a static public key seeks to obtain assurance of its owner's current possession of the corresponding static private key **shall** employ one of the following **approved** key-agreement schemes, incorporating key confirmation as specified in the indicated sections, with the recipient of that static public key acting as party U and serving as a key-confirmation recipient:

- dhHybridOneFlow (see Section 6.2.1.1, and either Section 6.2.1.5.2 or Section 6.2.1.5.3),
- (Cofactor) One-Pass Unified Model (see Section 6.2.1.2, and either Section 6.2.1.5.2 or Section 6.2.1.5.3),
- MQV1 (see Sections 6.2.1.3, and either Section 6.2.1.5.2 or Section 6.2.1.5.3),
- One-Pass MQV (see Sections 6.2.1.4, and either Section 6.2.1.5.2 or Section 6.2.1.5.3),
- dhOneFlow (see Sections 6.2.2.1 and 6.2.2.3.1), or
- (Cofactor) One-Pass Diffie-Hellman (see Sections 6.2.2.2 and 6.2.2.3.1).

5.6.2.2.4 Recipient Assurance of the Owner's Possession of an Ephemeral Private Key

This Recommendation does not require the recipient of an ephemeral public key to obtain assurance of the possession of the corresponding ephemeral private key by its claimed owner (i.e., the other participant in a key-establishment transaction). However, such assurance may be desired by the recipient, insisted upon by the recipient's organization, and/or required by a particular application.

Ephemeral key pairs are generated by their owner when needed (typically for a single use), and their private components are destroyed shortly thereafter (See Section 5.6.3.3 for details). Thus, the opportunity for the recipient of an ephemeral public key to obtain assurance that its claimed owner is in possession of the corresponding ephemeral private key is limited to the (single) key-establishment transaction during which it was received.

If all of the following conditions are met during a key-agreement transaction that incorporates key confirmation as specified in this Recommendation, then in the course of establishing keying material, the recipient of an ephemeral public key may be able to obtain assurance that the other participant (i.e., the claimed owner) is in possession of the corresponding ephemeral private key:

1) The recipient of the ephemeral public key also receives a static public key that is presumed to be owned by the other party and is used in the key-agreement transaction. (Therefore, the other party is the claimed owner of both the received static public key and the received ephemeral public key.)

2) The recipient of the ephemeral public key contributes its own (distinct) ephemeral public key to the key-agreement process, one that is intended to be arithmetically combined with the claimed owner's ephemeral private key in computations performed by the claimed owner. (If an appropriate key-agreement scheme is employed, the claimed owner will be challenged to demonstrate current knowledge of his ephemeral private key by successfully performing those computations during the transaction.)

3) The recipient of the ephemeral public key is also a key-confirmation recipient, with the claimed owner serving as the key-confirmation provider. (By successfully providing key confirmation, the claimed owner can demonstrate that he actually is the owner of the received static public key and that he knows the ephemeral private key corresponding to the received ephemeral public key.)

There are a limited number of key-agreement schemes specified in this Recommendation that can be used while satisfying all three of the conditions above. In order to claim conformance with this Recommendation, the key-agreement transaction during which the recipient of an ephemeral public key seeks to obtain assurance of the claimed owner's possession of the corresponding ephemeral private key **shall** employ one of the following **approved** key-agreement schemes, incorporating key confirmation as specified in the indicated sections, with the recipient of the ephemeral public key serving as a key-confirmation recipient:

- dhHybrid1 (see Section 6.1.1.1, and Section 6.1.1.5) or
- (Cofactor) Full Unified Model (see Sections 6.1.1.2 and 6.1.1.5).

Note: If key confirmation is provided in both directions in a key-agreement transaction employing one of the schemes above, then each party can obtain assurance of the other party's possession of their ephemeral private key.

5.6.2.3 Public Key Validation Routines

5.6.2.3.1 FFC Full Public-Key Validation Routine

FFC full public-key validation refers to the process of checking all the arithmetic properties of a 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 routine **shall** be used with static and/or ephemeral FFC public keys when assurance of the validity of such keys is to be obtained as specified in Sections 5.6.2.1.3, 5.6.2.2.1, or 5.6.2.2.2.

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 $2 \leq y \leq p\text{-}2$.

   (This step is to ensure that the public key has the unique correct representation and range for the field.)

2. Verify that $1 = y^q \bmod p$.

   (This step is to ensure that the public key has the correct order and thus is in the correct subgroup.)

Output: If any of the above verifications fail, then output an error indicator. Otherwise, output an indication of successful validation.

5.6.2.3.2 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 routine **shall** be used with static and/or ephemeral ECC public keys when assurance of the validity of such keys is to be obtained in accordance with the requirements of Section 5.6.2.1.3 or Section 5.6.2.2.1, or as specified in method 1 of Section 5.6.2.2.2.

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 $Ø$. This can be done by inspection if the point is entered in the standard affine representation.

   (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 is an odd prime p, 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 is an odd prime p, verify that $(y_Q)^2 = (x_Q)^3 + ax_Q + b$ in $GF(p)$, where the arithmetic is performed modulo p.

   If $q = 2^m$, verify that $(y_Q)^2 + x_Q\, y_Q = (x_Q)^3 + a(x_Q)^2 + b$ in $GF(2^m)$, where the arithmetic is performed as dictated by the field representation parameter FR.

   (This step is to ensure that the public key is on the correct elliptic curve.)

4. Verify that $nQ = Ø$.

   (This step is to ensure that the public key has the correct order. Along with the verification in step 1, ensures that the public key is in the correct range in the correct EC subgroup; that is, it is in the correct EC subgroup and is not the identity element $Ø$.)

Output: If any of the above verifications fail, then output an error indicator. Otherwise, output an indication of successful validation.

5.6.2.3.3 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 routine **shall** be used with ephemeral ECC public keys when assurance of the validity of such keys is to be obtained in accordance with method 2 of Section 5.6.2.2.2.

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 $Ø$. This can be done by inspection if the point is entered in the standard affine representation.

   (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 is an odd prime p, 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 is an odd prime p, verify that $(y_Q)^2 = (x_Q)^3 + ax_Q + b$ in *GF(p)*, where the arithmetic is performed modulo p.

   If $q = 2^m$, verify that $(y_Q)^2 + x_Q\, y_Q = (x_Q)^3 + a(x_Q)^2 + b$ in $GF(2^m)$, where the arithmetic is performed as dictated by the field representation parameter *FR*.

   (This step is to ensure that the public key is on the correct elliptic curve.)

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

Output: If any of the above verifications fail, then output an error indicator. Otherwise, output an indication of validation success.

5.6.3 Key Pair Management

5.6.3.1 Common Requirements on Static and Ephemeral Key Pairs

The following are common requirements on static and ephemeral key pairs (see the Recommendation for Key Management [7]):

1. Each private/public key pair **shall** be correctly associated with its corresponding specific set of domain parameters. A key pair **shall not** be used with more than one set of domain parameters.
2. Each key pair **shall** be generated as specified in Section 5.6.1.
3. Private keys **shall** be protected from unauthorized access, disclosure, modification and substitution.
4. Public keys **shall** be protected from unauthorized modification and substitution. This is often accomplished by using public-key certificates that have been signed by a Certification Authority (CA).

5.6.3.2 Specific Requirements on Static Key Pairs

The additional specific requirements for static key pairs are as follows:

1. The owner of a static key pair **shall** confirm the validity of the key pair by obtaining assurance of the correct generation of the key pair, private and public-key validity, and pair-wise consistency. The owner **shall** know the methods used to provide/obtain these assurances. (See Section 5.6.2.1 for further details.)
2. A recipient of a static public key **shall** be assured of the integrity and correct association of (a) the public key, (b) the set of domain parameters for that key, and (c) an identifier for the entity that owns the key pair (that is, the party with whom the recipient intends to

establish a key). This assurance is often provided by verifying a public-key certificate that was signed by a trusted third party (for example, a CA), but may be provided by direct distribution of the keying material from the owner, provided that the recipient trusts the owner to do this. See Section 4.1.

3. A recipient of a static public key **shall** obtain assurance of the validity of the public key. This assurance may be provided, for example, through the use of a public-key certificate if the CA obtains sufficient assurance of public-key validity as part of its certification process. See Section 5.6.2.2.1.

4. A recipient of a static public key **shall** have assurance of the owner's possession of the corresponding private key (see Section 5.6.2.2.3). The recipient **shall** know the method used to provide assurance to the recipient of the owner's possession of the private key. This assurance may be provided, for example, through the use of a public-key certificate if the CA obtains sufficient assurance of possession as part of its certification process.

5. 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 (for example, a digital signature key pair is not to be used for key establishment or vice versa) with the following possible exception: when requesting the (initial) certificate for a public static key-establishment key, the key-establishment private key associated with the public key may be used to sign the certificate request. See SP 800-57, Part 1 on Key Usage for further information.

5.6.3.3 Specific Requirements on Ephemeral Key Pairs

The additional specific requirements on ephemeral key pairs are as follows:

1. An ephemeral private key **shall** be used in exactly one key-establishment transaction, with one exception: an ephemeral private key may be used in multiple DLC key-transport transactions that are transporting identical secret keying material simultaneously (or within a short period of time; see the broadcast scenario in Section 7). After its use, an ephemeral private key **shall** be destroyed.

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. The owner of an ephemeral key pair **shall** confirm the validity of the key pair by obtaining assurance of correct generation, private- and public-key validity, and pair-wise consistency. The owner **shall** know the methods used to provide/obtain these assurances. These assurances can be obtained by virtue of the technique used by the owner to generate the key pair. (See Section 5.6.2.1 for further details.)

4. A recipient of an ephemeral public key **shall** have assurance of the validity of the public key (see Section 5.6.2.2.2).

5. If a recipient of an ephemeral public key requires assurance that the claimed owner of that public key has possession of the corresponding private key, then, in order to obtain that assurance in compliance with this Recommendation, such assurance **shall** be

obtained as specified in Section 5.6.2.2.4. Although other methods are sometimes used to provide such assurance, this Recommendation makes no statement as to their adequacy.

5.7 DLC Primitives

A primitive is a relatively simple operation that is defined to facilitate implementation in hardware or in a software subroutine. Each key-establishment scheme **shall** use exactly one DLC primitive. Each scheme in Section 6 **shall** use an appropriate primitive from the following list:

1. The FFC DH primitive (Section 5.7.1.1 of this Recommendation): 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 5.7.1.2 of this Recommendation and called the Modified Diffie-Hellman primitive in ANS 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 5.7.2.1 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 5.7.2.2 of this Recommendation): 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 output from these primitives **shall** be used as input to a key derivation method (see Section 5.8).

5.7.1 Diffie-Hellman Primitives

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

A 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 party U or party 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 $z = 1$, destroy all intermediate values used in the attempted computation of Z (including z) and output an error indicator.
3. Else, convert z to Z using the integer-to-byte-string conversion routine defined in Appendix C.1.
4. Destroy the results of all intermediate calculations used in the computation of Z (including z).
5. Output Z.

Output: The shared secret Z or an error indicator.

5.7.1.2 Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive

A 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 party U or party V.

Input:

1. (q, 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 $P = \emptyset$, destroy all intermediate values used in the attempted computation of Z (including z) and output an error indicator.
3. Else, convert $z = x_P$ to Z, using the field-element-to-byte string conversion routine defined in Appendix C.2, where x_P is the x-coordinate of P.
4. Destroy the results of all intermediate calculations used in the computation of Z (including z).
5. Output Z.

Output: The shared secret Z or an error indicator.

5.7.2 MQV Primitives

5.7.2.1 Finite Field Cryptography MQV (FFC MQV) Primitive

A 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 party U or party 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]
5. t_A : One's own second public key, and
6. t_B : The other party's second public key.

Process:

1. $w = \left\lceil \frac{1}{2} \log_2 q \right\rceil$.
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 $z = 1$, destroy all intermediate values used in the attempted computation of Z (including z) and output an error indicator.
7. Else, convert z to Z using the integer-to-byte-string conversion routine defined in Appendix C.1.
8. Destroy the results of all intermediate calculations used in the computation of Z (including z).
9. Output Z.

Output: The shared secret Z or an error indicator.

5.7.2.1.1 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 uses 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 party U or party V.

[5] In the FFC MQV primitive, a second key may be either ephemeral or static, depending on which form is being used, see Sections 5.7.2.1.1 and 5.7.2.1.2.

In this form, one's own second private and public keys (items 4 and 5 of the input list in Section 5.7.2.1) are one's own ephemeral private and public keys (r_A and t_A), and the other party's second public key (item 6 in Section 5.7.2.1) is the other party's ephemeral public key (t_B).

5.7.2.1.2 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, party U uses a static key pair and an ephemeral key pair, but party V uses only a static key pair. One-Pass MQV is done using the MQV primitive by using party V's static key pair as the second key pair (as party V has no ephemeral key pair).

Party U uses party V's static public key for the other party's second public key; that is, when party U uses the algorithm in Section 5.7.2.1, item 6 of the input list becomes party V's static public key (y_B).

Party V uses his/her static private key for his second private key; that is, when party V uses the algorithm in Section 5.7.2.1, input 4 of the input list becomes party V's static private key x_A, and item 5 becomes his static public key (y_A).

5.7.2.2 ECC MQV Associate Value Function

The associate value function is used by the ECC MQV family of key-agreement schemes to compute an integer that is associated with an elliptic curve point. This Recommendation defines avf(Q) to be the associate value function of a public key Q 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. Q: A public key (that is, Q is a point in the subgroup of order n and not equal to the point at infinity $Ø$).

Process:

1. Convert x_Q to an integer xqi using the convention specified in Appendix C.3.
2. Calculate

   $xqm = xqi \bmod 2^{\lceil f/2 \rceil}$ (where $f = \lceil \log_2 n \rceil$).

3. Calculate the associate value function

   avf(Q) = $xqm + 2^{\lceil f/2 \rceil}$. (See footnote[6]).

Output: avf(Q), the associate value of Q.

[6] Note that avf(Q) can be computed using only bit operations.

5.7.2.3 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. Assume that the party performing the computation is party A, and the other party is party B. Note that party A could be either party U or party 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,[7]
5. $Q_{e,A}$: One's own second public key, and
6. $Q_{e,B}$: The other party's second public key.

Process:

1. $implicitsig_A = (d_{e,A} + \text{avf}(Q_{e,A})d_{s,A}) \bmod n$.
2. $P = h(implicitsig_A)(Q_{e,B} + \text{avf}(Q_{e,B})Q_{s,B})$.
3. If $P = Ø$, destroy all intermediate values used in the attempted computation of Z (including z) and output an error indicator.
4. Else, convert $z=x_P$ to Z, using the field-element-to-byte-string conversion routine defined in Appendix C.2, where x_P is the x-coordinate of P.
5. Destroy the results of all intermediate calculations used in the computation of Z (including z).
6. Output Z.

Output: The shared secret Z or an error indicator.

5.7.2.3.1 Full MQV Form of the ECC MQV Primitive

This form of invoking the ECC 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 party U or party V.

In this form, one's own second private and public keys (item 4 and 5 of the input list in Section 5.7.2.3) are one's own ephemeral private and public keys ($d_{e,A}$ and $Q_{e,A}$), and the other party's

[7] In the ECC MQV primitive, a second key may be either ephemeral or static, depending on which form is being used, see Sections 5.7.2.3.1 and 5.7.2.3.2.

second public key (item 6 of the input list in Section 5.7.2.3) is the other party's ephemeral public key ($Q_{e,B}$).

5.7.2.3.2 One-Pass Form of the ECC 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, party U has a static key pair and an ephemeral key pair, but party V has only a static key pair. One-Pass MQV uses the MQV primitive with party V's static key pair as the second key pair (as party V has no ephemeral keys).

Party U uses party V's static public key as the other party's second public key. When party U uses the algorithm in Section 5.7.2.3, input 6 of the input list becomes party V's static public key ($Q_{s,B}$).

Party V uses his static private key as his second private key. When party V uses the algorithm in Section 5.7.2.3, item 4 of the input list becomes party V's static private key $d_{s,A}$, and item 5 becomes the his static public key ($Q_{s,A}$).

5.8 Key-Derivation Methods for Key Agreement Schemes

This section introduces **approved** key-derivation methods for use in key establishment as specified in this Recommendation. An **approved** key-derivation method **shall** be used to derive keying material from the shared secret, *Z*, that is computed during the execution of a key-agreement scheme specified in this Recommendation.

Key-derivation methods that conform to this Recommendation include the use of an **approved** single-step key-derivation function (KDF), as well as the use of an **approved** two-step (extraction-then-expansion) key-derivation procedure – see Sections 5.8.1 and 5.8.2, respectively. Additional **approved** application-specific key-derivation methods are discussed in Section 5.8.3. Other key-derivation methods may be temporarily allowed for backward compatibility; these other allowable methods – and any restrictions on their use – will be specified in [FIPS 140 IG].

When employed during the execution of a key-agreement scheme as specified in this Recommendation, the agreed-upon key-derivation method uses input that includes a freshly computed shared secret, *Z*. The derived keying material **shall** be computed in its entirety before outputting any portion of it, and (all copies of) *Z* **shall** be destroyed immediately following its use.

The output produced by a key-derivation method using input that includes the shared secret computed during the execution of any key-agreement scheme specified in this Recommendation **shall** only be used as secret keying material – such as a symmetric key used for data encryption or message integrity, a secret initialization vector, or, perhaps, a master (key derivation) key that will be used to generate additional keying material (possibly using a different process – see [SP 800-108]). Non-secret keying material (such as a non-secret initialization vector) **shall not** be generated from input that includes the shared secret.

5.8.1 The Single-step Key-Derivation Function

This section specifies an **approved** key-derivation function (KDF) that is executed in a single step, rather than the two-step procedure discussed in Section 5.8.2. The input to the KDF includes the shared secret *Z* (represented as a byte string).

This single-step KDF uses an auxiliary function H, which can be either 1) an **approved** hash function, denoted as *hash*, as defined in [FIPS 180] or 2) an HMAC with an **approved** hash function hash, denoted as HMAC-*hash*, as defined in [FIPS 198]. Table 6 and Table 7 identify the minimum output block length for the hash functions and HMACs required for each FFC and ECC parameter-size set.

Table 6: Minimum Output Length of H for FFC Domain Parameter-Size Sets

FFC Parameter Set Name	FA	FB	FC
Maximum security strength supported (in bits)	80	112	112
Bit length of field size p (i.e., $\lceil \log_2 p \rceil$)	1024	2048	2048
Bit length of subgroup order q (i.e., $\lceil \log_2 q \rceil$)	160	224	256
Minimum output length of H (in bits)	80	112	128

When an FFC parameter set is selected in a key-establishment scheme, any **approved** hash function, *hash,* can be used to define the auxiliary function, H, whether H = *hash* or H = HMAC-*hash* (see Section 5.8.1.1).

Table 7: Minimum Output Length of H for ECC Domain Parameter Sets

ECC Parameter Set Name	EA	EB	EC	ED	EE
Maximum security strength supported (in bits)	80	112	128	192	256
Bit length of ECC subgroup order n (i.e., $\lceil \log_2 n \rceil$)	160-223	224-255	256-383	384-511	512+
Minimum output length of H (in bits)	80	112	128	192	256

Regardless of the form chosen for the auxiliary function H (i.e., H = *hash* or H = HMAC *hash*), the **approved** hash function, *hash*, used to define H **shall** be selected as follows:

- For ECC parameter sets EA, EB and EC, any **approved** hash function can be used as *hash*.
- For parameter sets ED and EE, any **approved** hash function other than SHA-1 can be used as *hash*; SHA-1 **shall not** be used, since the output block length of SHA-1 is only 160 bits.

- For parameter set EE, any **approved** hash function with an output block length greater than or equal to 256 bits can be used as *hash*; hash functions with output block lengths less than 256 bits (e.g., SHA-224 and SHA-512/224) **shall not** be used.

5.8.1.1 The Single-Step KDF Specification

This section specifies an **approved** single-step key-derivation function (KDF), whose input includes the shared secret *Z* (represented as a byte string) and other information.

The KDF is specified as follows:

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

where *OtherInput* consists of *keydatalen* and *OtherInfo*.

Auxiliary Function H (two options):

Option 1: H(*x*) = *hash*(*x*), where *hash* is an **approved** hash function (see Section 5.1) meeting the selection requirements specified in this Recommendation (see Sections 5.5.1 and 5.8.1), and the input, *x*, is a bit string.

Option 2: H(*x*) = HMAC-*hash*(*salt*, *x*), where HMAC-*hash* is an instantiation of the HMAC function (as defined in [FIPS 198]) employing an **approved** hash function, *hash* (see Section 5.1), and *hash* meets the selection requirements specified in this Recommendation (see Sections 5.5.1 and 5.8.1). An implementation-dependent byte string, *salt*, serves as the HMAC key, and *x* (the input to H) is a bit string that serves as the HMAC "message" – as specified in [FIPS 198].

Implementation-Dependent Parameters:

1. *hashlen*: an integer that indicates the length (in bits) of the output block of the hash function, *hash*, employed by the auxiliary function, H, that is used to derive blocks of secret keying material.
2. *max_H_inputlen*: an integer that indicates the maximum permitted length (in bits) of the bit string, *x*, that is used as input to the auxiliary function, H.
3. *salt*: a (public or private) byte string that is only required when an HMAC-based auxiliary function is implemented (see Option 2 above). The *salt* could be, for example, a value computed from nonces exchanged as part of a key-establishment protocol that employs one or more of the key-agreement schemes specified in this Recommendation, a value already shared by the protocol participants, or a value that is pre-determined by the protocol. In this case, the length of the *salt* can be any agreed-upon length. However, if there is no means of selecting the *salt*, then it **shall** be an all-zero byte string whose bit length equals that specified as the length of the input block for the hash function, *hash*.

Input:

1. *Z*: a byte string that represents the shared secret z.
2. *keydatalen*: An integer that indicates the length (in bits) of the secret keying material to be derived; *keydatalen* **shall** be less than or equal to $hashlen \times (2^{32} - 1)$.
3. *OtherInfo*: A bit string of context-specific data (see Section 5.8.1.2 for details).

Process:

1. *reps* = $\lceil$ *keydatalen* / *hashlen* $\rceil$.
2. If *reps* > $(2^{32} - 1)$, then ABORT: return an error indicator.
3. Initialize a 32-bit, big-endian bit string *counter* as 00000001_{16} (i.e. 0x00000001).
4. If *counter* || *Z* || *OtherInfo* is more than *max_H_inputlen* bits long, then ABORT: return an error indicator.
5. For *i* = 1 to *reps* by 1, do the following:
   5.1 Compute *K*(*i*) = H(*counter* || *Z* || *OtherInfo*).
   5.2 Increment *counter* (modulo 2^{32}), treating it as an unsigned 32-bit integer.
6. Let *K_Last* be set to *K*(*reps*) if (*keydatalen* / *hashlen*) is an integer; otherwise, let *K_Last* be set to the (*keydatalen* mod *hashlen*) leftmost bits of *K*(*reps*).
7. Set *DerivedKeyingMaterial* = *K*(*1*) || *K*(2) || … || *K*(*reps-1*) || *K_Last*.

Output:

The bit string *DerivedKeyingMaterial* of length *keydatalen* bits (or an error indicator).

Notes:

When an **approved** key-agreement scheme is used to determine a shared secret *Z*, the participants **should** know which entity is acting as "party U" and which entity is acting as "party V" to ensure (among other things) that they will derive the same keying material. (See Section 6 for descriptions of the specific actions required of parties U and V during the execution of each of the **approved** key-agreement schemes.) The roles of party U and V **shall** be assigned to the key-establishment participants by the protocol employing the key-agreement scheme.

In step 5.1 above, the entire output of the hash function *hash* **shall** be used whether H(*x*) = *hash*(*x*) or H(*x*) = HMAC-*hash*(*salt*, *x*). Therefore, the bit length of each output block of H is *hashlen* bits. Some of the hash functions specified in [FIPS 180] are defined with an internal truncation operation (e.g., SHA-384). In these cases, the "entire output" of *hash* is the output value as defined in [FIPS 180] (e.g., for SHA-384, the entire output is defined to be the 384 bits resulting from the internal truncation, so *hashlen* = 384, in this case). Any truncation performed by the KDF (external to *hash*) is done in step 6.

5.8.1.2 *OtherInfo*

The bit string *OtherInfo* **should** be used to ensure that the derived keying material is adequately "bound" to the context of the key agreement transaction. Although other methods may be used to bind keying material to the transaction context, this Recommendation makes no statement as to the adequacy of these other methods. Failure to adequately bind the derived keying material to the transaction context could adversely affect the types of assurance that can be provided by certain key agreement schemes.

Context-specific information that may be appropriate for inclusion in *OtherInfo*:

- Public information about parties U and V, such as their identifiers.
- The public keys contributed by each party to the key agreement transaction. (In the case of a static public key, one could include a certificate that contains the public key.)
- Other public and/or private information shared between parties U and V before or during the transaction, such as nonces or pre-shared secrets.
- An indication of the protocol or application employing the key derivation method.
- Protocol-related information, such as a label or session identifier.
- The desired length of the derived keying material.
- An indication of the key-agreement scheme and/or key-derivation method used.
- An indication of the domain parameters associated with the asymmetric key pairs employed for key establishment.
- An indication of other parameter or primitive choices (e.g., hash functions, MacTag lengths, etc.).
- An indication of how the derived keying material should be parsed, including an indication of which algorithm(s) will use the (parsed) keying material.

For rationale in support of including entity identifiers, scheme identifiers, and/or other information in *OtherInfo*, see Appendix B.

The meaning of each information item and each item's position within the bit string *OtherInfo* **shall** be specified. In addition, each item of information included in *OtherInfo* **shall** be unambiguously represented, for example, as a fixed-length bit string or in the form *Datalen* || *Data*, where *Data* is a variable-length string of zero or more (eight-bit) bytes, and *Datalen* is a fixed-length, big-endian counter that indicates the length (in bytes) of *Data*. These requirements can be satisfied, for example, by using ASN.1 DER encoding as specified in 5.8.1.2.2 for *OtherInfo*.

Recommended formats for *OtherInfo* are specified in Sections 5.8.1.2.1 and 5.8.1.2.2. One of these two formats **should** be used by the single-step KDF specified in Section 5.8.1.1 when the auxiliary function employed is H = *hash*. When the recommended formats are used, the included items of information **shall** be divided into (three, four, or five) subfields as defined below.

AlgorithmID: A required subfield that indicates how the derived keying material will be parsed and for which algorithm(s) the derived secret keying material will be used. For example, *AlgorithmID* might indicate that bits 1-112 are to be used as a 112-bit HMAC key and that bits 113-240 are to be used as a 128-bit AES key.

PartyUInfo: A required subfield containing public information about party U. At a minimum, *PartyUInfo* **shall** include ID_U, an identifier for party U, as a distinct item of information. This subfield could also include information about the public key(s) contributed to the key agreement transaction by party U. The nonce provided by party U as required in a C(0e, 2s) scheme (see Section 6.3) **shall** be included in this subfield.

PartyVInfo: A required subfield containing public information about party V. At a minimum, *PartyVInfo* **shall** include ID_V, an identifier for party V, as a distinct item of information. This subfield could also include information about the public key(s) contributed to the key agreement transaction by party V. The nonce provided by party V when acting as a key-

confirmation recipient in a C(1e, 2s) scheme or a C(0e, 2s) scheme **should** be included in this field (see Sections 6.2.1.5 and 6.3.3).

SuppPubInfo: An optional subfield containing additional, mutually known public information (e.g., *keydatalen*, the domain parameters associated with the keys used to derive the shared secret, an identifier for the particular key-agreement scheme that was used to form *Z*, an indication of the protocol or application employing that scheme, a session identifier, etc. This is particularly useful if these aspects of the key-agreement transaction can vary – see Appendix B for further discussion).

SuppPrivInfo: An optional subfield containing additional, mutually known private information (e.g., a shared secret symmetric key that has been communicated through a separate channel).

5.8.1.2.1 The Concatenation Format for *OtherInfo*

This section specifies the concatenation format for *OtherInfo*. This format has been designed to provide a simple means of binding the derived keying material to the context of the key-agreement transaction, independent of other actions taken by the relying application. Note: When the single-step KDF specified in Section 5.8.1.1 is used with H = *hash* as the auxiliary function and this concatenation format for *OtherInfo*, the resulting key-derivation method is the Concatenation Key Derivation Function specified in the original version of SP 800-56A.

For this format, *OtherInfo* is a bit string equal to the following concatenation:

AlgorithmID || *PartyUInfo* || *PartyVInfo* {|| *SuppPubInfo* }{|| *SuppPrivInfo* },

where the five subfields are bit strings comprised of items of information as described in Section 5.8.1.2.

Each of the three required subfields *AlgorithmID*, *PartyUInfo*, and *PartyVInfo* **shall** be the concatenation of a pre-determined sequence of substrings in which each substring represents a distinct item of information. Each such substring **shall** have one of these two formats: either it is a fixed-length bit string, or it has the form *Datalen* || *Data* – where *Data* is a variable-length string of zero or more (eight-bit) bytes, and *Datalen* is a fixed-length, big-endian counter that indicates the length (in bytes) of *Data*. (In this variable-length format, a null string of data **shall** be represented by a zero value for *Datalen*, indicating the absence of following data.) A protocol using this format for *OtherInfo* **shall** specify the number, ordering and meaning of the information-bearing substrings that are included in each of the subfields *AlgorithmID*, *PartyUInfo*, and *PartyVInfo*, and **shall** also specify which of the two formats (fixed-length or variable-length) is used by each such substring to represent its distinct item of information. The protocol **shall** specify the lengths for all fixed-length quantities, including the *Datalen* counters.

Each of the optional subfields *SuppPrivInfo* and *SuppPubInfo* (when allowed by the protocol employing the one-step KDF) **shall** be the concatenation of a pre-determined sequence of substrings representing additional items of information that may be used during key derivation upon mutual agreement of parties U and V. Each substring representing an item of information **shall** be of the form *Datalen* || *Data*, where *Data* is a variable-length string of zero or more (eight-bit) bytes and *Datalen* is a fixed-length, big-endian counter that indicates the length (in bytes) of *Data*; the use of this form for the information allows U and V to omit a particular

information item without confusion about the meaning of the other information that is provided in the *SuppPrivInfo* or *SuppPubInfo* subfield. The substrings representing items of information that parties *U* and *V* choose not to contribute are set equal to *Null*, and are represented in this variable-length format by setting *Datalen* equal to zero. If a protocol allows the use of the *OtherInfo* subfield *SuppPrivInfo* and/or the subfield *SuppPubInfo*, then the protocol **shall** specify the number, ordering and meaning of additional items of information that may be used in the allowed subfield(s) and **shall** specify the fixed-length of the *Datalen* counters.

5.8.1.2.2 The ASN.1 Format for *OtherInfo*

The ASN.1 format for *OtherInfo* provides an alternative means of binding the derived keying material to the context of the key-agreement transaction, independent of other actions taken by the relying application. Note: When the single-step KDF specified in Section 5.8.1.1 is used with H = *hash* as the auxiliary function and this ASN.1 format for *OtherInfo*, the resulting key-derivation method is the ASN.1 Key Derivation Function specified in the original version of SP 800-56A.

For the ASN.1 format, *OtherInfo* is a bit string resulting from the ASN.1 DER encoding (see [ISO/IEC 8825-1]) of a data structure comprised of a sequence of three required subfields *AlgorithmID*, *PartyUInfo*, and *PartyVInfo*, and, optionally, a subfield *SuppPubInfo* and/or a subfield *SuppPrivInfo* – as described in Section 5.8.1.2. A protocol using this format for *OtherInfo* **shall** specify the type, ordering and number of distinct items of information included in each of the (three, four, or five) subfields employed.

5.8.1.2.3 Other Formats for *OtherInfo*

Formats other than those provided in Sections 5.8.1.2.1 and 5.8.1.2.2 (e.g., those providing the items of information in a different arrangement) may be used for *OtherInfo*, but the context-specific information described in the preceding sections **should** be included (see the discussion in Section 5.8.1.2). This Recommendation makes no statement as to the adequacy of other formats.

5.8.2 The Extraction-then-Expansion Key-Derivation Procedure

This Recommendation permits the use of an **approved** extraction-then-expansion key-derivation procedure as an alternative to the single-step key-derivation function specified in Section 5.8.1. When the extraction-then-expansion key-derivation procedure is employed in an **approved** key-agreement scheme, the secret keying material is derived in two steps. The first step is called (randomness) extraction, and the second step is called (key) expansion. In the extraction step, the input is the shared secret *Z* (represented as a byte string) along with a salt. The output of the extraction step is a key-derivation key. In the expansion step, the key-derivation key is used with additional data, such as that included in the *OtherInfo* used by the KDFs specified above; also see Appendix B for guidance) to derive the keying material with required length. The details of the **approved** extraction-then-expansion key-derivation procedure are specified in [SP 800-56C].

5.8.3 Application-Specific Key-Derivation Methods

Additional **approved** application-specific key-derivation methods are enumerated in [SP 800-135]. Unless an explicit exception is made in that document, any hash function employed by the

key-derivation methods enumerated in [SP 800-135] **shall** be **approved** and **shall** also meet the selection requirements specified in this Recommendation for the auxiliary function H (see Sections 5.5.1 and 5.8.1).

5.9 Key Confirmation

The term *key confirmation* (KC) refers to actions taken to provide assurance to one party (the key-confirmation *recipient*) that another party (the key-confirmation *provider*) is in possession of a (supposedly) shared secret and/or confirm that the other party has the correct version of keying material that was derived or transported during a key-establishment transaction. (Correct, that is, from the perspective of the key-confirmation recipient.) Such actions are said to provide *unilateral key confirmation* when they provide this assurance to only one of the participants in the key-establishment transaction; the actions are said to provide *bilateral key confirmation* when this assurance is provided to both participants (i.e., when unilateral key confirmation is provided in both directions).

Oftentimes, key confirmation is obtained (at least implicitly) by some means external to the key-establishment scheme employed during a transaction (e.g., by using a symmetric key that was established during the transaction to decrypt an encrypted message sent later by the key-confirmation provider), but this is not always the case. In some circumstances, it may be appropriate to incorporate the exchange of explicit key-confirmation information as an integral part of the key-establishment scheme itself. The inclusion of key confirmation may enhance the security services that can be offered by a key-establishment scheme. For example, when certain key-agreement schemes incorporate key confirmation (as described in this Recommendation), they can be used to provide the recipient with assurance that the provider is in possession of the private key corresponding to a particular public key, from which the recipient may infer that the provider is the owner of that key pair (see Sections 5.6.2.2.3 and 5.6.2.2.4).

For key confirmation to comply with this Recommendation, key confirmation **shall** be incorporated into an **approved** key-establishment scheme as specified in the sections that follow.

5.9.1 Unilateral Key Confirmation for Key-Agreement Schemes

As specified in this Recommendation, unilateral key confirmation occurs when one participant in the execution of a key-agreement scheme (the key-confirmation "provider") demonstrates to the satisfaction of the other participant (the key-confirmation "recipient") that both the provider and the recipient have possession of the same secret *MacKey*.

MacKey is a symmetric key derived using the (shared) secret *Z* that was computed by each party during that particular execution of the key-agreement scheme (see Section 5.8 for key derivation). *MacKey* and certain context-specific *MacData* (as specified in Sections 5.9.1.1) are used by the provider as input to an **approved** MAC algorithm to obtain a *MacTag* that is sent to the recipient. The recipient performs an independent computation of the *MacTag*. If the *MacTag* value computed by the key-confirmation recipient matches the *MacTag* value received from the key-confirmation provider, then key confirmation is successful. (See Section 5.2 for *MacTag* generation and verification, and Section 5.9.3 for MAC security discussion.)

Successful key confirmation provides assurance to the recipient that the same *Z* value has been computed by both parties and that the two parties have used *Z* in the same way to derive shared keying material.

Unilateral key confirmation is an optional feature that can be incorporated into any key-agreement scheme in which the key-confirmation provider is required to own a static key-establishment key pair that is used in the key-establishment process. If the intended key-confirmation recipient is not required to contribute an ephemeral public key to the key-establishment process, then the recipient **shall** instead contribute a nonce that is used as part of the input to the key-derivation method employed by the scheme. Each party is required to have an identifier, chosen in accordance with the assumptions stated for the key-agreement scheme.

5.9.1.1 Adding Unilateral Key Confirmation to a Key-Agreement Scheme

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. Additional details will be provided for each scheme in the appropriate subsections of Section 6. In the discussion that follows, the key-confirmation provider, P, may be either party U or party V, as long as P has a static key pair. The key-confirmation recipient, R, is the other party.

1. If the recipient, R, is not required to generate an ephemeral key pair as part of the key-agreement scheme, then R **shall** contribute a random nonce to be used (in addition to the shared secret *Z*) as input to the key-derivation method employed by the scheme; see Section 5.4 for a discussion of the length and security strength required for the nonce.

2. The provider, P, computes

   $$MacData_P = message\_string_P \| ID_P \| ID_R \| EphemData_P \| EphemData_R \{\| Text_P\}$$

   where

   - $message\_string_P$ is a six byte string with a value of "KC_1_U" when party U is providing the *MacTag*, or "KC_1_V" when party V is providing the *MacTag*. (Note that these values will be changed for bilateral key confirmation, as specified in Section 5.9.2.)
   - ID_P is the identifier used to label the key-confirmation provider.
   - ID_R is the identifier used to label the key confirmation recipient.
   - $EphemData_P$ and $EphemData_R$ are ephemeral values (corresponding to ephemeral public keys or nonces) contributed by the provider and recipient, respectively. Their precise values are specified in the subsections of Section 6 that describe how key confirmation can be incorporated into the particular schemes included in this Recommendation. $EphemData_P$ is null only in the case that the provider has contributed neither an ephemeral public key nor a nonce during the scheme. For example, in a C(1e, 2s) scheme with unilateral key confirmation from party V to party U as introduced in Section 5.2.1.5.2, party V only contributes a static key pair; in this case, $EphemData_V$ can be null.

- $Text_P$ is an optional bit string that may be used during key confirmation and that is known by both parties.

3. After computing the shared secret *Z* and applying the key-derivation method to obtain *DerivedKeyingMaterial* (see Section 5.8), the provider parses *DerivedKeyingMaterial* into two parts, *MacKey* and *KeyData*:

   MacKey || *KeyData* = *DerivedKeyingMaterial.*

4. The provider computes $MacTag_P$ (see Sections 5.2.1 and 5.9.3):

   $MacTag_P = T_{MacLen}$[MAC (*MacKey*, , $MacData_P$)],

   and sends it to the recipient.

5. The recipient forms $MacData_P$, determines *MacKey*, computes $MacTag_P$ 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. Thus, the recipient also has assurance that the provider could compute *KeyData* correctly.

Both parties **shall** destroy the *MacKey* once it is no longer needed to provide or obtain key confirmation.

If, during a particular key-agreement transaction, it happens that $MacTag_P$ cannot be verified by the recipient, then key confirmation has failed and all of the derived keying material (*MacKey* and *KeyData*) **shall** be destroyed by each participant. In particular, *DerivedKeyingMaterial* **shall not** be revealed by either participant to any other party (not even to the other participant), and the derived keying material **shall not** be used for any further purpose. In the case of a key-confirmation failure, the key-agreement transaction **shall** be discontinued.

Unilateral key confirmation may be added in either direction to any of the C(2e, 2s), C(1e, 2s) and C(0e, 2s) schemes; it may also be added to the C(1e, 1s) schemes, but only when party V (the party contributing the static key pair) is the key confirmation provider, and party U is the key confirmation recipient. (See the relevant subsections of Section 6.)

5.9.2 Bilateral Key Confirmation for Key-Agreement Schemes

Bilateral key confirmation is an optional feature that can be incorporated into any key-agreement scheme in which each party is required to own a static key-establishment key pair that is used in the key-establishment process. Bilateral key confirmation is accomplished by performing unilateral key confirmation in both directions (with U providing $MacTag_U$ to recipient V, and V providing $MacTag_V$ to recipient U) during the same scheme. If a party is not also required to contribute an ephemeral public key to the key-establishment process, then that party **shall** instead contribute a random nonce that is used as part of the input to the key-derivation method employed by the scheme; see Section 5.4 for a discussion of the length and security strength

required for the nonce. Each party is required to have an identifier, chosen in accordance with the assumptions stated for the key-agreement scheme.

5.9.2.1 Adding Bilateral Key Confirmation to a Key-Agreement Scheme

To include bilateral key confirmation, two instances of unilateral key confirmation (as specified in Section 5.9.1.1, subject to the modifications listed below) **shall** be incorporated into the scheme, once with party U as the key-confirmation provider (i.e., P = U and R = V) and once with party V as the provider (i.e., P = V and R = U). Additional details will be provided for each scheme in the appropriate subsections of Section 6.

In addition to setting P = U and R = V in one instance of the unilateral key-confirmation procedure described in Section 5.9.1.1 and setting P = V and R = U in a second instance, the following changes/clarifications apply when using the procedure for bilateral key confirmation:

1. When computing $MacTag_U$, the value of the six-byte $message\_string_U$ that forms the initial segment of $MacData_U$ is "KC_2_U".
2. When computing $MacTag_V$, the value of the six-byte $message\_string_V$ that forms the initial segment of $MacData_V$ is "KC_2_V".
3. If used at all, the value of the (optional) byte string $Text_U$ used to form the final segment of $MacData_U$ can be different than the value of the (optional) byte string $Text_V$ used to form the final segment of $MacData_V$, provided that both parties are aware of the value(s) used.

Bilateral key confirmation may be added to the C(2e, 2s), C(1e, 2s) and C(0e, 2s) schemes, as specified in the relevant subsections of Section 6.

5.9.3 Security Strength of the Mac Tag

In this Recommendation, a *MagTag* used for key confirmation **shall** be generated using an **approved** MAC algorithm, which can be an HMAC [FIPS 198] with an **approved** hash function or a CMAC [SP 800-38B] with an **approved** block cipheher (see Section 5.2). The domain parameter set **shall** be used to determine the minimum length of the key (*MacKey* length) used to compute the *MacTag* and the minimum length of the *MacTag*, the (possibly truncated) result from the MAC computation using the MAC algorithm with *MacKey*, as shown in Table 8 and Table 9 for the FFC and ECC domain parameter sets, respectively.

Table 8: Minimum *MacKey* Length and *MacLen* for FFC Domain Parameter Sets

FFC Parameter Set Name	FA	FB	FC
Maximum security strength supported (in bits)	80	112	112
Bit length of the field size p (i.e., $\lceil \log_2 p \rceil$)	1024	2048	2048

FFC Parameter Set Name	FA	FB	FC
Bit length of the subgroup order q (i.e., $\lceil \log_2 q \rceil$)	160	224	256
Minimum *MacKey* length (in bits)	80	112	128
Minimum *MacLen*, i.e., the *MacTag* length (in bits)	80	112	128

HMAC using any **approved** hash function or AES CMAC with any **approved** AES key length (128, 192 or 256 bits) can be used with all the FFC domain parameter sets described in Table 8.

Table 9: Minimum *MacKey* Length and *MacLen* for ECC Domain Parameter Sets

ECC Parameter Set Name	EA	EB	EC	ED	EE
Maximum security strength supported (in bits)	80	112	128	192	256
Bit length of ECC the subgroup order n (i.e., $\lceil \log_2 n \rceil$)	160-223	224-255	256-383	384-511	512+
Minimum *MacKey* length (in bits)	80	112	128	192	256
Minimum *MacLen*, i.e., the *MacTag* length (in bits)	80	112	128	192	256

As shown in Table 9, when an HMAC is used for key confirmation for the ECC schemes, any **approved** hash function can be used for the ECC domain parameter sets EA, EB, and EC; however, HMAC with the SHA-1 hash function **shall not** be used for domain parameter sets ED and EE, because the maximum length of the *MacTag* that an HMAC with SHA-1 can generate is only 160 bits. For the same reason, an HMAC with SHA-224 and SHA-512/224 **shall not** be used for key confirmation with ECC parameter set EE. AES CMAC can be used for key confirmation with any AES key length (128, 192, 256 bits) for the ECC domain parameter sets EA, EB, and EC; however, AES CMAC **shall not** be used for domain parameter sets ED and EE, because the maximum length of the *MacTag* that an AES CMAC can generate is only 128 bits, the AES output block length.

6. Key Agreement

This Recommendation provides three categories of key-agreement schemes (see Table 10). 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*e), parties U and V have a total of *i* ephemeral key pairs. The first category, C(2e), consists of schemes requiring the generation of ephemeral key pairs by both parties; a C(2e) scheme is suitable for an interactive key establishment protocol. The second category, C(1e), consists of schemes requiring the generation of an ephemeral key pair by only one party; a C(1e) scheme is suitable for a store and forward

scenario, but may also be used in an interactive key establishment protocol. The third category, C(0e), consists of schemes that do not use ephemeral keys.

Key confirmation may be added to many of these schemes to provide assurance that the participants share the same keying material; see Section 5.9 for details on key confirmation. Each party **should** have such assurance. Although other methods are often used to provide this assurance, this Recommendation makes no statement as to the adequacy of these other methods.

Table 10: Key-agreement Scheme Categories

Category	Comment
C(2e): Two ephemeral key pairs	Each party generates an ephemeral key pair.
C(1e): One ephemeral key pair	Only party U generates an ephemeral key pair.
C(0e): Zero ephemeral key pairs	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 11). In subcategory C(*ie, js*), parties U and V have a total of *i* ephemeral key pairs and *j* static key pairs. The suitability for interactive or store-and-forward protocols of each subcategory is discussed in Section 8.

Table 11: Key-agreement scheme subcategories

Category	Subcategory
C(2e): Two ephemeral key pairs	C(2e, 2s): Each party generates an ephemeral key pair and uses a static key pair.
	C(2e, 0s): Each party generates an ephemeral key pair; no static key pairs are used.
C(1e): One ephemeral key pair	C(1e, 2s): Party U generates an ephemeral key pair and uses a static key pair; party V uses only a static key pair.
	C(1e, 1s): Party U generates an ephemeral key pair, but uses no static key pair; party V uses only a static key pair.
C(0e): Zero ephemeral key pairs	C(0e, 2s): Each party uses only a static key pair.

The schemes may be further classified by whether they use finite field cryptography (FFC) or elliptic curve cryptography (ECC). A scheme may use either Diffie-Hellman or MQV primitives (see Section 5.7). Thus, for example, notation C(2e, 2s, FFC DH) completely classifies the dhHybrid1 scheme of Section 6.1.1.1 as a scheme with two ephemeral keys and two static keys that uses finite field cryptography and a Diffie-Hellman primitive (see Table 12). The names of these schemes are taken from ANS X9.42 and ANS X9.63.

Table 12: Key-agreement schemes

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

Each party in a key-agreement process **shall** use the same set of valid domain parameters. These parameters **shall** be established, and assurance of their validity **shall** be obtained prior to the generation of key pairs and the initiation of the key-agreement process. See Section 5.5 for a discussion of domain parameters.

If Party U uses a static key pair in a given key-agreement transaction, then Party U **shall** have an identifier, ID_U, that has an association with the static key pair that is known (or discoverable) and trusted by party V (i.e., there **shall** be a trusted association between ID_U and party U's static

public key). If party U does not contribute a static public key as part of a key-agreement transaction, then ID_U (if required for that transaction) is a non-null identifier selected in accordance with the relying application/protocol. Similar rules apply to Party V's identifier, ID_V.

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 or nonces that occur during the key-agreement or key-confirmation process. Note that the flow diagrams 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 error conditions are handled in the process text.

For each scheme, there are conditions that must be satisfied to enable proper use of that scheme. These conditions are listed as the *assumptions*. Failure to meet all such conditions could yield undesirable results, such as the inability to communicate or the loss of security. As part of the proper implementation of this Recommendation, system users and/or agents trusted to act on their behalf (including application developers, system installers, and system administrators) are responsible for ensuring that all assumptions are satisfied at the time a key-establishment transaction takes place.

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

In this category, each party generates an ephemeral key pair and sends the ephemeral public key to the other party. This category consists of two subcategories that are determined by the use of static keys by the parties. In the first subcategory, each party contributes both static and ephemeral keys (see Section 6.1.1), while in the second subcategory, each party contributes only ephemeral keys (see Section 6.1.2).

6.1.1 C(2e, 2s) Schemes

Figure 4 depicts a typical flow for a C(2e, 2s). For these schemes, each party (U and V) contributes 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 secret keying material is derived from the shared secret.

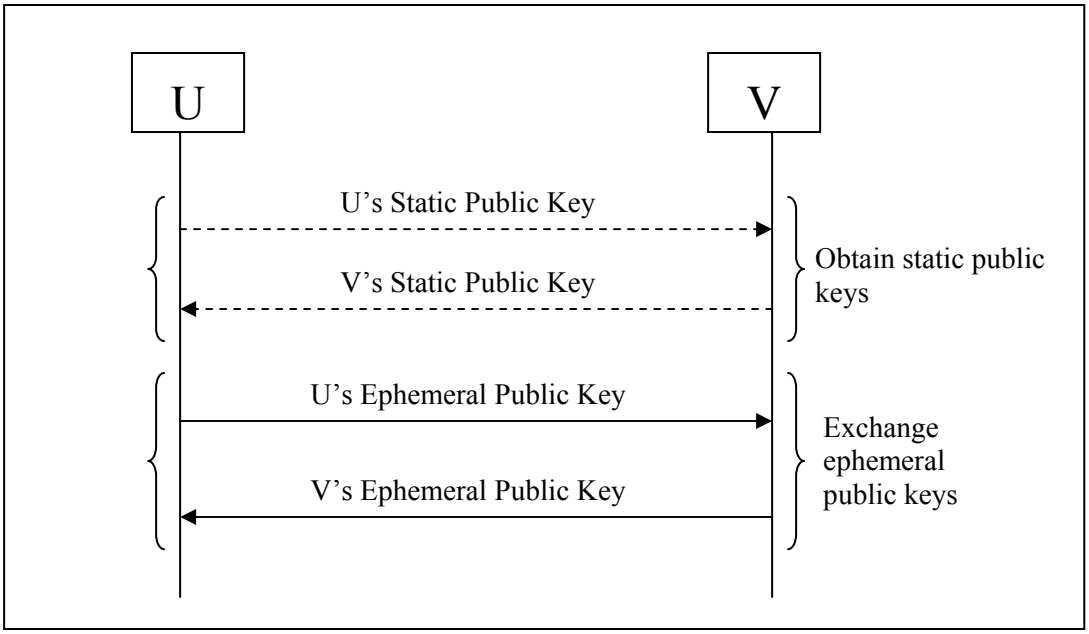

Figure 4: C(2e, 2s) schemes: Each party contributes a static and an ephemeral key pair

Assumptions: In order to execute a C(2e, 2s) key-establishment scheme in compliance with this Recommendation, the following assumptions **shall be** true.

1. Each party has an authentic copy of the same set of domain parameters, *D*, that have been generated as specified in Section 5.5.1. For FFC schemes, $D = (p, q, g\{, SEED, pgenCounter\})$; for ECC schemes, $D = (q, FR, a, b\{, SEED\}, G, n, h)$. Furthermore, each party has obtained assurance of the validity of these domain parameters as specified in Section 5.5.2.

2. Each party has been designated as the owner of a static key pair that was generated as specified in Section 5.6.1 using the set of domain parameters, *D*. For FFC schemes, the static key pair is (x, y); for ECC schemes, the static key pair is (d_s, Q_s). Each party has obtained assurance of the validity of its own static public key as specified in Section 5.6.2.1.3 and has obtained assurance of its possession of the correct value for its own private key as specified in Section 5.6.2.1.5.

3. The parties have agreed upon an **approved** key-derivation method (see Section 5.8), as well as other associated parameters related to cryptographic elements to be used (see Section 5). If key confirmation is used, the parties have also agreed upon an **approved** MAC and associated parameters (see Table 8 and Table 9 in Section 5.9.3).

4. Prior to or during the key-agreement process, each party receives the other party's static public key in a trusted manner (e.g., from a certificate signed by a trusted CA or directly from the other party, who is trusted by the recipient). Each party has obtained assurance of the validity of the other party's static public key as specified in Section 5.6.2.2.

5. The recipient of a static public key has obtained assurance that its (claimed) owner is (or was) in possession of the corresponding static private key, as specified in Section 5.6.2.2.3.

6. When an identifier is used to label a party during the key agreement process, that identifier has a trusted association to that party's static public key. (In other words, whenever both the identifier and static public key of one participant are employed in the key-agreement process, they are associated in a manner that is trusted by the other participant.) When an identifier is used to label a party during the key-agreement process, both parties are aware of the particular identifier employed for that purpose.

6.1.1.1 dhHybrid1, C(2e, 2s, FFC DH) Scheme

This section describes the dhHybrid1 scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.1.1 are true. In particular, it is assumed that party U has obtained the static public key y_V of party V, and party V has obtained the static public key y_U of party U.

With the exception of key derivation, dhHybrid1 is "symmetric" in the actions of parties U and V. Only the actions performed by party U are specified here; a specification of the actions performed by party V may be obtained by systematically replacing the letter "U" by "V" (and vice versa) in the description of the key-agreement transformation. Note, however, that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair (r_U, t_U) from the domain parameters *D* as specified in Section 5.6.1.1. Send the public key t_U to V. Receive an ephemeral public key t_V (purportedly) from V. If t_V is not received, destroy the ephemeral private key r_U, and return an error indicator without performing the remaining actions.

2. Verify that t_V is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, destroy the ephemeral private key r_U, and return an error indicator without performing the remaining actions.

3. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret Z_s from the set of domain parameters *D*, U's static private key x_U, and V's static public key y_V. If the call to the FFC DH primitive outputs an error indicator, destroy the ephemeral private key r_U, destroy the results of all intermediate calculations used in the attempted computation of Z_s, and return an error indicator without performing the remaining actions.

4. Use the FCC DH primitive to derive a shared secret Z_e from the set of domain parameters *D*, U's ephemeral private key r_U, and V's ephemeral public key t_V. If this call to the FFC DH primitive outputs an error indicator, destroy Z_s and the ephemeral private key r_U, destroy the results of all intermediate calculations used in the attempted computation of Z_e, and return an error indicator without performing the remaining actions.

5. Compute the shared secret $Z = Z_e \| Z_s$. Destroy Z_e and Z_s.

6. Use the agreed-upon key derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key r_U, and return an error indicator without performing the remaining actions.
7. If the ephemeral private key r_U will not be used in a broadcast scenario for subsequent key-establishment transactions using this scheme (see Section 7), then destroy r_U.
8. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note 1: Key confirmation can be incorporated into this scheme. See Section 6.1.1.5 for details.

Note 2: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 7 above).

dhHybrid1 is summarized in Table 13.

Table 13: dhHybrid1 Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	*D* = (*p*, *q*, *g*{, *SEED*, *pgenCounter*})	*D* = (*p*, *q*, *g*{, *SEED*, *pgenCounter*})
Static Data	Static private key x_U Static public key y_U	Static private key x_V Static public key y_V
Ephemeral Data	Ephemeral private key r_U Ephemeral public key t_U	Ephemeral private key r_V Ephemeral public key t_V
Computation	1. Compute Z_s by calling FFC DH using x_U and y_V 2. Compute Z_e by calling FFC DH using r_U and t_V 3. Compute $Z = Z_e \| Z_s$	1. Compute Z_s by calling FFC DH using x_V and y_U 2. Compute Z_e by calling FFC DH using r_V and t_U 3. Compute $Z = Z_e \| Z_s$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

6.1.1.2 (Cofactor) Full Unified Model, C(2e, 2s, ECC CDH) Scheme

This section describes the Full Unified Model scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.1.1 are true. In particular, it is assumed that party U has obtained the static public key $Q_{s,V}$ of party V, and party V has obtained the static public key $Q_{s,U}$ of party U.

With the exception of key derivation, the Full Unified Model is "symmetric" in the actions of parties U and V. Only the actions performed by party U are specified here; a specification of the actions performed by party V may be obtained by systematically replacing the letter "U" by "V" (and vice versa) in the description of the key-agreement transformation. Note, however, that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) from the domain parameters *D* as specified in Section 5.6.1.2. Send the public key $Q_{e,U}$ to V. Receive an ephemeral public key $Q_{e,V}$ (purportedly) from V. If $Q_{e,V}$ is not received, destroy the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
2. Verify that $Q_{e,V}$ is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, destroy the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions..
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret Z_s from the set of domain parameters *D*, U's static private key $d_{s,U}$, and V's static public key $Q_{s,V}$. If the call to the ECC CDH primitive outputs an error indicator, destroy the ephemeral private key $d_{e,U}$, destroy the results of all intermediate calculations used in the attempted computation of Z_s, and return an error indicator without performing the remaining actions.
4. Use the ECC CDH primitive to derive a shared secret Z_e from the set of domain parameters D, U's ephemeral private key $d_{e,U}$, and V's ephemeral public key $Q_{e,V}$. If this call to the ECC CDH primitive outputs an error indicator, destroy Z_s and the ephemeral private key $d_{e,U}$, destroy the results of all intermediate calculations used in the attempted computation of Z_e, and return an error indicator without performing the remaining actions.
5. Compute the shared secret $Z = Z_e \| Z_s$. Destroy Z_e and Z_s.
6. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
7. If the ephemeral private key $d_{e,U}$ will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy $d_{e,U}$.
8. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note 1: Key confirmation can be incorporated into this scheme. See Section 6.1.1.5 for details.

Note 2: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 7 above).

The Full Unified Model is summarized in Table 14.

Table 14: Full Unified Model Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	$D = (q, FR, a, b\{, SEED\}, G, n, h)$	$D = (q, FR, a, b\{, SEED\}, G, n, h)$
Static Data	Static private key $d_{s,U}$ Static public key $Q_{s,U}$	Static private key $d_{s,V}$ Static public key $Q_{s,V}$
Ephemeral Data	Ephemeral private key $d_{e,U}$ Ephemeral public key $Q_{e,U}$	Ephemeral private key $d_{e,V}$ Ephemeral public key $Q_{e,V}$
Computation	1. Compute Z_s by calling ECC CDH using $d_{s,U}$ and $Q_{s,V}$ 2. Compute Z_e by calling ECC CDH using $d_{e,U}$ and $Q_{e,V}$ 3. Compute $Z = Z_e \| Z_s$	1. Compute Z_s by calling ECC CDH using $d_{s,V}$ and $Q_{s,U}$ 2. Compute Z_e by calling ECC CDH using $d_{e,V}$ and $Q_{e,U}$ 3. Compute $Z = Z_e \| Z_s$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy Z	1. Compute *DerivedKeyingMaterial* 2. Destroy Z

6.1.1.3 MQV2, C(2e, 2s, FFC MQV) Scheme

This section describes the MQV2 scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.1.1 are true. In particular, it is assumed that party U has obtained the static public key y_V of party V, and party V has obtained the static public key y_U of party U.

With the exception of key derivation, MQV2 is "symmetric" in the actions of parties U and V. Only the actions performed by party U are specified here; a specification of the actions performed by party V may be obtained by systematically replacing the letter "U" by "V" (and vice versa) in the description of the key-agreement transformation. Note, however, that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair (r_U, t_U) from the domain parameters *D* as specified in Section 5.6.1.1. Send the public key t_U to V. Receive an ephemeral public key t_V (purportedly) from V. If t_V is not received, destroy the ephemeral private key r_U, and return an error indicator without performing the remaining actions.
2. Verify that t_V is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, destroy the ephemeral private key r_U, and return an error indicator without performing the remaining actions.
3. Use the MQV2 form of the FFC MQV primitive in Section 5.7.2.1 to derive a shared secret *Z* from the set of domain parameters *D*, U's static private key x_U, V's static public key y_V, U's ephemeral private key r_U, U's ephemeral public key t_U, and V's ephemeral public key t_V. If the call to the FFC MQV primitive outputs an error indicator, destroy the ephemeral private key r_U, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key r_U, and return an error indicator without performing the remaining actions.
5. If the ephemeral private key t_U will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy r_U.
6. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note 1: Key confirmation can be incorporated into this scheme. See Section 6.1.1.5 for details.

Note 2: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 5 above).

MQV2 is summarized in Table 15.

Table 15: MQV2 Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	*D* =(*p, q, g*{, *SEED, pgenCounter*})	*D* =(*p, q, g*{, *SEED, pgenCounter*})

Static Data	Static private key x_U Static public key y_U	Static private key x_V Static public key y_V
Ephemeral Data	Ephemeral private key r_U Ephemeral public key t_U	Ephemeral private key r_V Ephemeral public key t_V
Computation	Compute Z by calling FFC MQV using x_U, y_V, r_U, t_U, and t_V	Compute Z by calling FFC MQV using x_V, y_U, r_V, t_V, and t_U
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy Z	1. Compute *DerivedKeyingMaterial* 2. Destroy Z

6.1.1.4 Full MQV, C(2e, 2s, ECC MQV) Scheme

This section describes the Full MQV scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.1.1 are true. In particular, it is assumed that party U has obtained the static public key $Q_{s,V}$ of party V, and party V has obtained the static public key $Q_{s,U}$ of party U.

With the exception of key derivation, the Full MQV scheme is "symmetric" in the actions of parties U and V. Only the actions performed by party U are specified here; a specification of the actions performed by party V may be obtained by systematically replacing the letter "U" by "V" (and vice versa) in the description of the key-agreement transformation. Note, however, that U and V must use identical orderings of the bit strings that are input to the key derivation function.

Party *U* **shall** execute the following transformation to a) establish a shared secret value Z with party V, and b) derive secret keying material from Z.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) from the domain parameters D as specified in Section 5.6.1.2. Send the public key $Q_{e,U}$ to V. Receive an ephemeral public key $Q_{e,V}$ (purportedly) from V. If $Q_{e,V}$ is not received, destroy the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
2. Verify that $Q_{e,V}$ is a valid public key for the parameters D as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, destroy the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
3. Use the Full MQV form of the ECC MQV primitive in Section 5.7.2.3.1 to derive a shared secret value Z from the set of domain parameters D, U's static private key $d_{s,U}$, V's static public key $Q_{s,V}$, U's ephemeral private key $d_{e,U}$, U's ephemeral public key $Q_{e,U}$, and V's ephemeral public key $Q_{e,V}$. If the call to the ECC MQV primitive outputs an error indicator, destroy the ephemeral private key $d_{e,U}$, destroy the results of all intermediate calculations used in the attempted computation of Z, and return an error indicator without performing the remaining actions.

4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
5. If the ephemeral private key $Q_{e,U}$ will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy $d_{e,U}$.
6. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note 1: Key confirmation can be incorporated into this scheme. See Section 6.1.1.5 for details.

Note 2: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 7 above).

The Full MQV is summarized in Table 16.

Table 16: Full MQV Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	*D* = (*q*, *FR*, *a*, *b*{, *SEED*}, *G*, *n*, *h*)	*D* = (*q*, *FR*, *a*, *b*{, *SEED*}, *G*, *n*, *h*)
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 $Q_{e,V}$
Computation	Compute *Z* by calling ECC MQV using $d_{s,U}$, $Q_{s,V}$, $d_{e,U}$, $Q_{e,U}$, and $Q_{e,V}$	Compute *Z* by calling ECC MQV using $d_{s,V}$, $Q_{s,U}$, $d_{e,V}$, $Q_{e,V}$, and $Q_{e,U}$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

6.1.1.5 Incorporating Key Confirmation into a C(2e, 2s) Scheme

The subsections that follow illustrate how to incorporate key confirmation (as described in Section 5.9) into the C(2e, 2s) key-agreement schemes described above.

The flow depictions separate the key-establishment flow from the key-confirmation flow. The depictions and accompanying discussions presume that the assumptions of the scheme have been satisfied, that the key-agreement transaction has proceeded successfully through key derivation, and that the received *MacTags* are successfully verified as specified in Section 5.2.2.

6.1.1.5.1 C(2e, 2s) Scheme with Unilateral Key Confirmation Provided by U to V

Figure 5 depicts a typical flow for a C(2e, 2s) scheme with unilateral key confirmation from party U to party V. In this scenario, party U and party V assume the roles of key-confirmation provider and recipient, respectively. The successful completion of this process provides party V with a) assurance that party U has derived the same secret *Z* value; and b) assurance that party U has actively participated in the process.

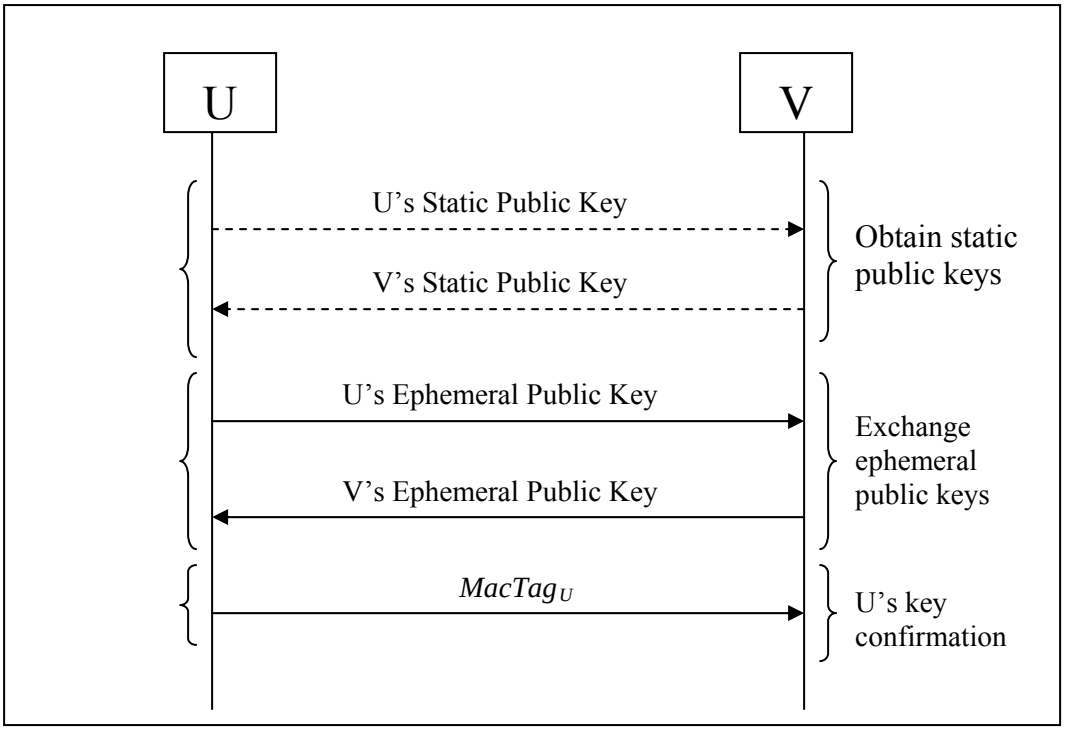

Figure 5: C(2e, 2s) scheme with unilateral key confirmation from U to V

To provide (and receive) key confirmation (as described in Section 5.9.1.1), U (and V) set

$$EphemData_U = EphemPubKey_U, \text{ and } EphemData_V = EphemPubKey_V:$$

Party U provides $MacTag_U$ to V (as specified in Section 5.9.1.1, with $P = U$ and $R = V$), where $MacTag_U$ is computed (as specified in Section 5.2.1) using

$$MacData_U = \text{“KC\_1\_U”} \| ID_U \| ID_V \| EphemPubKey_U \| EphemPubKey_V \{\| Text_U\}.$$

Party V (the key-confirmation recipient) uses the same format for $MacData_U$ to compute its own version of $MacTag_U$, and then verifies that the newly computed $MacTag_U$ matches the value provided by U.

6.1.1.5.2 C(2e, 2s) Scheme with Unilateral Key Confirmation Provided by V to U

Figure 6 depicts a typical flow for a C(2e, 2s) scheme with unilateral key confirmation from party V to party U. In this scenario, party V and party U assume the roles of key-confirmation provider and recipient, respectively. The successful completion of the key-confirmation process provides party U with a) assurance that party V has derived the same secret *Z* value; and b) assurance that party V has actively participated in the process.

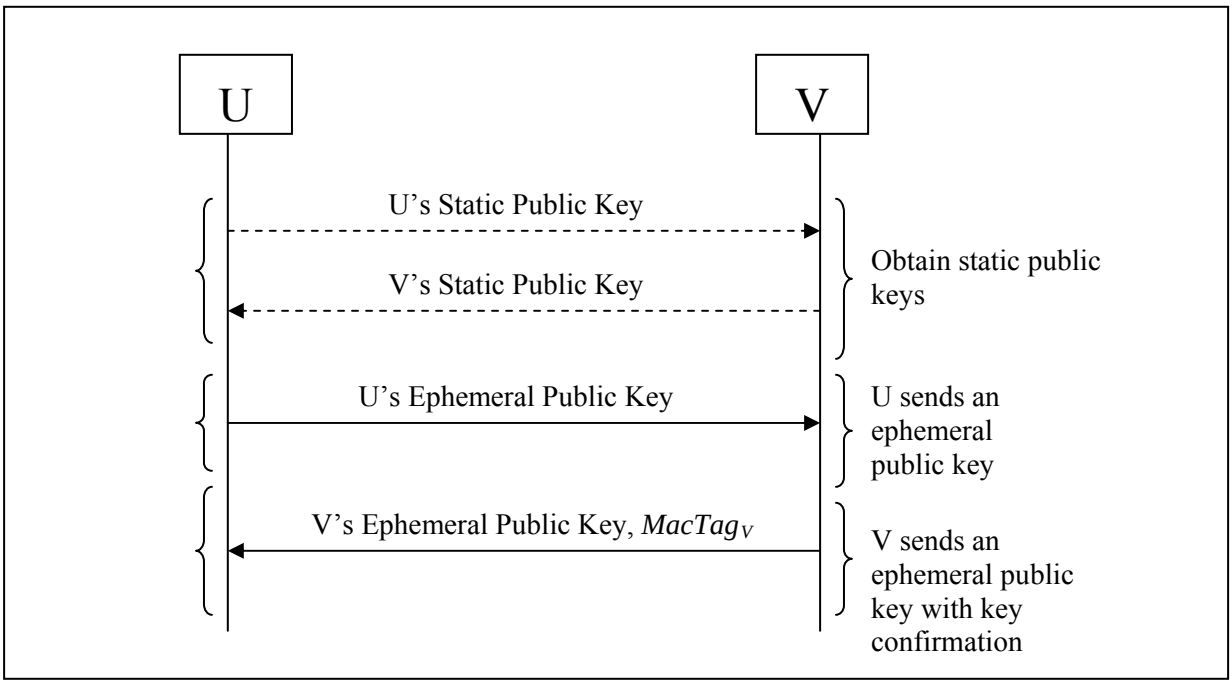

Figure 6: C(2e, 2s) scheme with unilateral key confirmation from V to U

To provide (and receive) key confirmation (as described in Section 5.9.1.1), V (and U) set

$EphemData_V = EphemPubKey_V$, and $EphemData_U = EphemPubKey_U$:

Party V provides $MacTag_V$ to U (as specified in Section 5.9.1.1, with $P = V$ and $R = U$), where $MacTag_V$ is computed (as specified in Section 5.2.1) using

$MacData_V$ = "KC_1_V" || ID_V || ID_U || $EphemPubKey_V$ || $EphemPubKey_U$ {|| $Text_V$}.

Party U (the key-confirmation recipient) uses the same format for $MacData_V$ to compute its own version of $MacTag_V$ and then verifies that the newly computed $MacTag_V$ matches the value provided by V.

Note that in Figure 6, party V's ephemeral public key ($EphemPubKey_V$) and the *MacTag* ($MacTag_V$) are depicted as being sent in the same message (to reduce the number of passes in the combined key-agreement/key-confirmation process). They may also be sent separately.

6.1.1.5.3 C(2e, 2s) Scheme with Bilateral Key Confirmation

Figure 7 depicts a typical flow for a C(2e, 2s) scheme with bilateral key confirmation. In this method, party U and party V assume the roles of both the provider and the recipient in order to

obtain bilateral key confirmation. The successful completion of the key-confirmation process provides each party with a) assurance that the other party has derived the same secret *Z* value, and b) also provides each party with assurance that the other party has actively participated in the process.

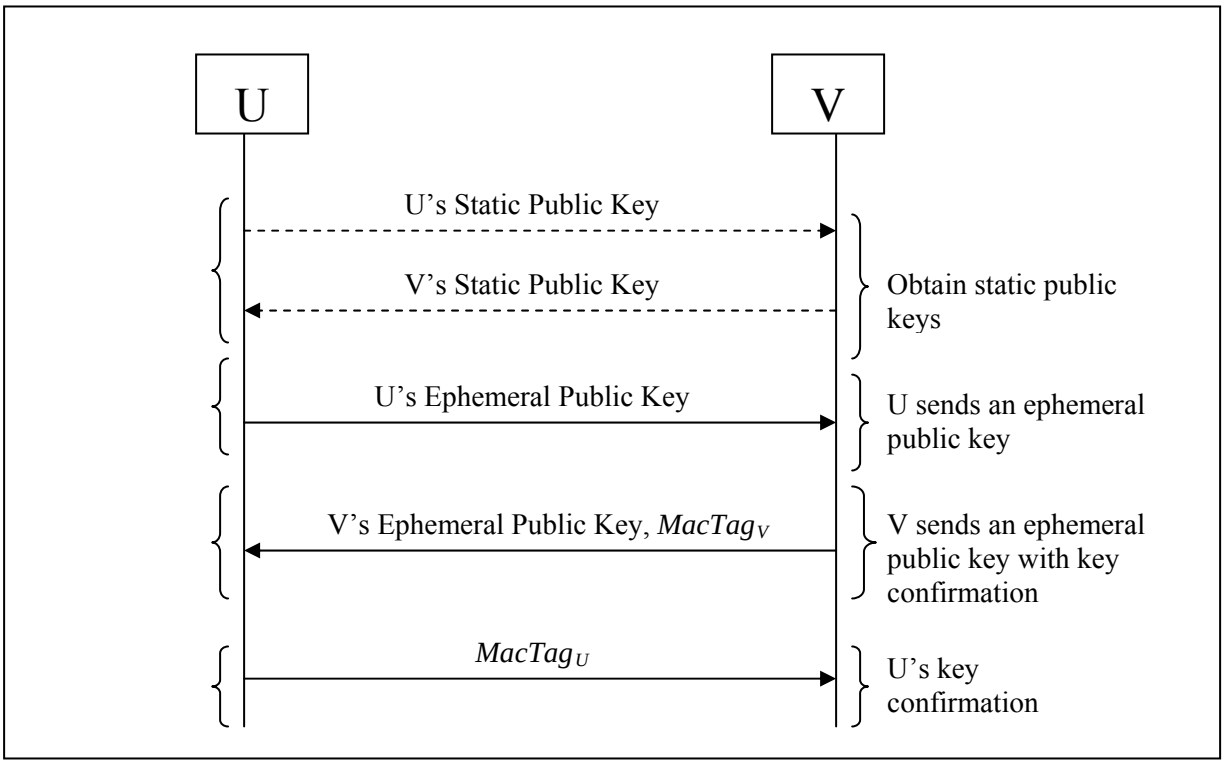

Figure 7: C(2e, 2s) scheme with bilateral key confirmation

To provide bilateral key confirmation (as described in Section 5.9.2.1), U and V exchange and verify *MacTags* that have been computed (as specified in Sections 5.2.1) using

$EphemData_U = EphemPubKey_U$, and $EphemData_V = EphemPubKey_V$:

Party V provides $MacTag_V$ to U (as specified in Sections 5.9.1.1 and 5.9.2.1, with $P = V$ and $R = U$); $MacTag_V$ is computed by V (and verified by U) using

$MacData_V$ = "KC_2_V" || ID_V || ID_U || $EphemPubKey_V$ || $EphemPubKey_U$ {|| $Text_V$}.

Party U provides $MacTag_U$ to V (as specified in Sections 5.9.1.1 and 5.9.2.1, with $P = U$ and $R = V$); $MacTag_U$ is computed by U (and verified by V) using

$MacData_U$ = "KC_2_U" || ID_U || ID_V || $EphemPubKey_U$ || $EphemPubKey_V$ {|| $Text_U$}.

Note that in Figure 7, party V's ephemeral public key ($EphemPubKey_V$) and the *MacTag* ($MacTag_V$) are depicted as being sent in the same message (to reduce the number of passes in the combined key-agreement/key-confirmation process). They may also be sent separately, and if sent separately, then the order in which the *MacTags* are sent could be reversed.

6.1.2 C(2e, 0s) Schemes

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 secret keying material is derived using the shared secret (see Figure 8).

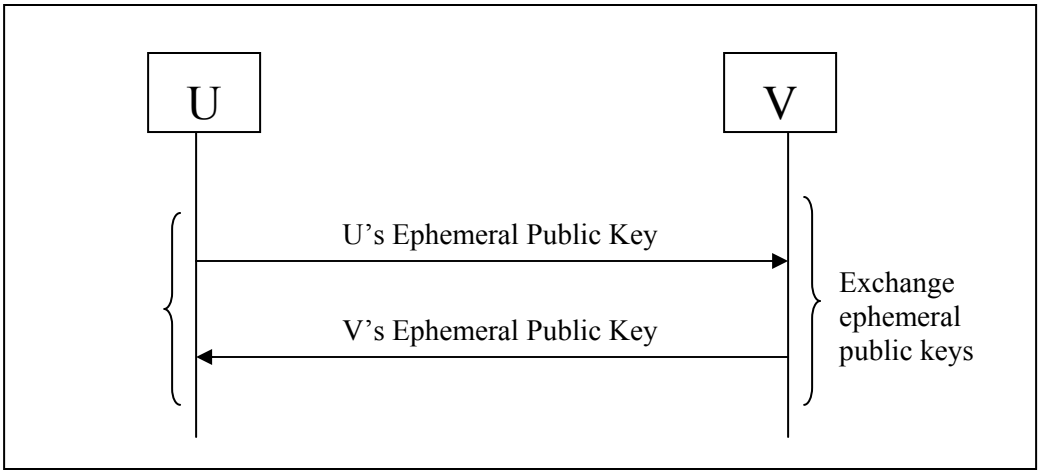

Figure 8: C(2e, 0s) schemes: each party contributes only an ephemeral key pair

Assumptions: In order to execute a C(2e, 0s) key-establishment scheme in compliance with this Recommendation, the following assumptions **shall be** true.

1. Each party has an authentic copy of the same set of domain parameters, *D*. These parameters have been generated as specified in Section 5.5.1. For FFC schemes, $D = (p, q, g\{, SEED, pgenCounter\})$; for ECC schemes, $D = (q, FR, a, b\{, SEED\}, G, n, h)$. Furthermore, each party has obtained assurance of the validity of these domain parameters as specified in Section 5.5.2.
2. The parties have agreed upon an **approved** key-derivation method, as well as an **approved** algorithm to be used with that method (e.g., a hash function) and other associated parameters to be used (see Section 5.8).
3. When an identifier is used to label a party during the key-agreement process, it has been selected/assigned in accordance with the requirements of the protocol relying upon the use of the key-agreement scheme, and its value is known to both parties.

6.1.2.1 dhEphem, C(2e, 0s, FFC DH) Scheme

This section describes the dhEphem scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.1.2 are true.

With the exception of key derivation, dhEphem is "symmetric" in the actions of parties U and V. Only the actions performed by party U are specified here; a specification of the actions performed by party V may be obtained by systematically replacing the letter "U" by "V" (and vice versa) in the description of the key-agreement transformation. Note, however, that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair (r_U, t_U) from the domain parameters *D* as specified in Section 5.6.1.1. Send the public key t_U to V. Receive an ephemeral public key t_V (purportedly) from V. If t_V is not received, destroy the ephemeral private key r_U, and return an error indicator without performing the remaining actions.
2. Verify that t_V is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, destroy the ephemeral key r_U), and return an error indicator without performing the remaining actions.
3. Use the FCC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* from the set of domain parameters *D*, U's ephemeral private key r_U, and V's ephemeral public key t_V. Then destroy the ephemeral private key r_U. If the call to the FFC DH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining action.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

dhEphem is summarized in Table 17.

Table 17: dhEphem Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	(*p, q, g*{, *SEED, pgenCounter*})	(*p, q, g*{, *SEED, pgenCounter*})
Static Data	N/A	N/A
Ephemeral Data	Ephemeral private key r_U Ephemeral public key t_U	Ephemeral private key r_V Ephemeral public key t_V
Computation	Compute *Z* by calling FFC DH using r_U and t_V	Compute *Z* by calling FFC DH using r_V and t_U

	Party U	Party V
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

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

This section describes the Ephemeral Unified Model scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.1.2 are true.

With the exception of key derivation, Ephemeral Unified Model is "symmetric" in the actions of parties U and V. Only the actions performed by party U are specified here; a specification of the actions performed by party V may be obtained by systematically replacing the letter "U" by "V" (and vice versa) in the description of the key-agreement transformation. Note, however, that U and V must use identical orderings of the bit strings that are input to the key derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) from the domain parameters *D* as specified in Section 5.6.1.2. Send the public key $Q_{e,U}$ to V. Receive an ephemeral public key $Q_{e,V}$ (purportedly) from V. If $Q_{e,V}$ is not received, destroy the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
2. Verify that $Q_{e,V}$ is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, destroy the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* from the set of domain parameters *D*, U's ephemeral private key $d_{e,U}$, and V's ephemeral public key $Q_{e,V}$. Then destroy the ephemeral private key $d_{e,U}$. If the call to the ECC CDH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining action.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

The Ephemeral Unified Model is summarized in Table 18.

Table 18: Ephemeral Unified Model Key-agreement Scheme

	Party U	Party V
Domain Parameters	(*q, FR, a, b*{, *SEED*}, *G, n, h*)	(*q, FR, a, b*{, *SEED*}, *G, n, h*)
Static Data	N/A	N/A
Ephemeral Data	Ephemeral private key $d_{e,U}$ Ephemeral public key $Q_{e,U}$	Ephemeral private key $d_{e,V}$ Ephemeral public key $Q_{e,V}$
Computation	Compute *Z* by calling ECC CDH using $d_{e,U}$ and $Q_{e,V}$	Compute *Z* by calling ECC CDH using $d_{e,V}$ and $Q_{e,U}$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

6.1.2.3 Key Confirmation for C(2e, 0s)

In a C(2e, 0s) key-agreement scheme, none of the parties contributes a static key pair. Only ephemeral key pairs are used to derive the secret value *Z*. Without a trusted association with an identifier of either party, key confirmation cannot achieve the expected purposes. Therefore, in this Recommendation, key confirmation is not incorporated for the C(2e, 0s) key-agreement schemes.

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

This category consists of two subcategories that are determined by the use (or non-use) of a static key pair by each of the parties. Only party U generates an ephemeral key pair. In the first subcategory, both party U and party V use a static key pair, and party U also generates an ephemeral key pair (see Section 6.2.1). In the second subcategory, party U generates an ephemeral key pair, but uses no static key pair; party V uses only a static key pair (see Section 6.2.2).

6.2.1 C(1e, 2s) Schemes

For these schemes, party U uses both static and ephemeral private/public key pairs. Party V 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 secret keying material is derived using the shared secret (see Figure 9).

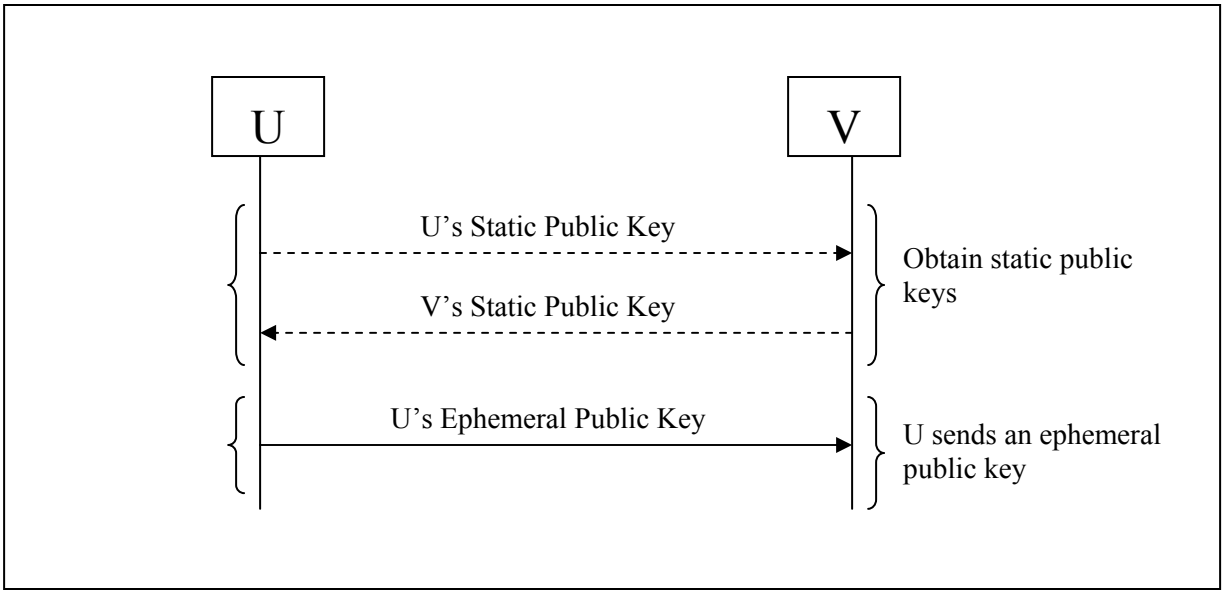

Figure 9: C(1e, 2s) schemes: U contributes a static and an ephemeral key pair while V contributes only a static key pair

Assumptions: In order to execute a C(1e, 2s) key-establishment scheme in compliance with this Recommendation, the following assumptions **shall be** true.

1. Each party has an authentic copy of the same set of domain parameters, *D*. These parameters have been generated as specified in Section 5.5.1. For FFC schemes, $D = (p, q, g\{, SEED, pgenCounter\})$; for ECC schemes, $D = (q, FR, a, b\{, SEED\}, G, n, h)$. Furthermore, each party has obtained assurance of the validity of these domain parameters as specified in Section 5.5.2.

2. Each party has been designated as the owner of a static key pair that was generated as specified in Section 5.6.1 using the set of domain parameters, *D*. For FFC schemes, the static key pair is (x, y); for ECC schemes, the static key pair is (d_s, Q_s). Each party has obtained assurance of the validity of its own static public key as specified in Section 5.6.2.1. Each party has also obtained assurance of its possession of the correct value for its own private key as specified in Section 5.6.3.1.

3. The parties have agreed upon an **approved** key-derivation method, as well as an **approved** algorithm to be used with that method (e.g., a hash function) and other associated parameters to be used for key derivation (see Section 5.8).

4. If key confirmation is used, the parties have also agreed upon an **approved** MAC and associated parameters (see Table 8 and Table 9 in Section 5.9.3).

5. Prior to or during the key-agreement process, each party receives the other party's static public key in a trusted manner (e.g., from a certificate signed by a trusted CA or directly from the other party, who is trusted by the recipient). Each party has obtained assurance of the validity of the other party's static public key as specified in Section 5.6.2.2.1.

6. The recipient of a static public key has obtained assurance that its (claimed) owner is (or was) in possession of the corresponding static private key, as specified in Section 5.6.2.2.3.

7. When an identifier is used to label a party during the key-agreement process, that identifier has a trusted association to that party's static public key. (In other words, whenever both the identifier and static public key of one participant are employed in the key-agreement process, they are associated in a manner that is trusted by the other participant.) When an identifier is used to label a party during the key-agreement process, both parties are aware of the particular identifier employed for that purpose.

6.2.1.1 dhHybridOneFlow, C(1e, 2s, FFC DH) Scheme

This section describes the dhHybridOneFlow scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.2.1 are true. In particular, it is assumed that party U has obtained the static public key y_V of party V, and party V has obtained the static public key y_U of party U.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair (r_U, t_U) from the domain parameters *D* as specified in Section 5.6.1.1. Send the public key t_U to V.

2. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret Z_s from the set of domain parameters *D*, U's static private key x_U, and V's static public key y_V. If the call to the FFC DH primitive outputs an error indicator, destroy the ephemeral private key r_U, destroy the results of all intermediate calculations used in the attempted computation of Z_s, and return an error indicator without performing the remaining actions.

3. Use the FCC DH primitive to derive a shared secret Z_e from the set of domain parameters *D*, U's ephemeral private key r_U, and V's static public key y_V. If this call to the FFC DH primitive outputs an error indicator, destroy Z_s and the ephemeral private key r_U,, destroy the results of all intermediate calculations used in the attempted computation of Z_e, and return an error indicator without performing the remaining actions.

4. Compute the shared secret $Z = Z_e \,||\, Z_s$. Destroy Z_e and Z_s.

5. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key r_U,, and return an error indicator without performing the remaining actions.

6. If the ephemeral private key r_U will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy r_U.
7. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 6 above).

Party V **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party U, and b) derive secret keying material from *Z*.

Actions: V derives secret keying material as follows:

1. Receive an ephemeral public key t_U (purportedly) from U. If t_U is not received, return an error indicator without performing the remaining actions.
2. Verify that t_U is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, return an error indicator without performing the remaining actions.
3. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret value Z_s from the set of domain parameters *D*, V's static private key x_V, and U's static public key y_U. If the call to the FFC DH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of Z_s, and return an error indicator without performing the remaining actions.
4. Use the FCC DH primitive to derive a shared secret Z_e from the set of domain parameters *D*, V's static private key x_V, and U's ephemeral public key t_U. If this call to the FFC DH primitive outputs an error indicator, destroy Z_s, destroy the results of all intermediate calculations used in the attempted computation of Z_e, and return an error indicator without performing the remaining actions.
5. Compute the shared secret $Z = Z_e \,||\, Z_s$. Destroy Z_e and Z_s.
6. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining action.
7. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation can be incorporated into this scheme. See Section 6.2.1.5 for details.

dhHybridOneFlow is summarized in Table 19.

Table 19: dhHybridOneFlow Key-agreement Scheme Summary

	Party U	Party V
Domain Parameters	(p, q, g{, *SEED*, *pgenCounter*})	(p, q, g{, *SEED*, *pgenCounter*})
Static Data	Static private key x_U Static public key y_U	Static private key x_V Static public key y_V
Ephemeral Data	Ephemeral private key r_U Ephemeral public key t_U	N/A
Computation	1. Compute Z_s by calling FFC DH using x_U and y_V 2. Compute Z_e by calling FFC DH using r_U and y_V 3. Compute $Z = Z_e \| Z_s$	1. Compute Z_s by calling FFC DH using x_V and y_U 2. Compute Z_e by calling FFC DH using x_V and t_U 3. Compute $Z = Z_e \| Z_s$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy Z	1. Compute *DerivedKeyingMaterial* 2. Destroy Z

6.2.1.2 (Cofactor) One-Pass Unified Model, C(1e, 2s, ECC CDH) Scheme

This section describes the One-Pass Unified Model scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.2.1 are true. In particular, it is assumed that party U has obtained the static public key $Q_{s,V}$ of party V, and party V has obtained the static public key $Q_{s,U}$ of party U.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value Z with party V, and b) derive secret keying material from Z.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) from the domain parameters D as specified in Section 5.6.1.2. Send the public key $Q_{e,U}$ to V.
2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret Z_s from the set of domain parameters D, U's static private key $d_{s,U,}$ and V's static public key $Q_{s,V}$**.** If the call to the ECC CDH primitive outputs an error indicator, destroy the ephemeral private key $d_{e,U,}$, destroy the results of all intermediate calculations used in the attempted computation of Z_s, and return an error indicator without performing the remaining actions..

3. Use the ECC CDH primitive to derive a shared secret Z_e – another element of the finite field of size q – from the set of domain parameters D, U's ephemeral private key $d_{e,U,}$ and V's static public key $Q_{s,V.}$ If this call to the ECC CDH primitive outputs an error indicator, destroy Z_s and the ephemeral private key $d_{e,U,}$, destroy the results of all intermediate calculations used in the attempted computation of Z_e, and return an error indicator without performing the remaining actions.
4. Compute the shared secret $Z = Z_e \,||\, Z_{s.}$ Destroy Z_e and Z_s.
5. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value Z and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of Z and the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
6. If the ephemeral private key $d_{e,U}$ will not be used in a broadcast scenario (see Section 7) for subsequent key establishment transactions using this scheme, then destroy $d_{e,U}$.
7. Destroy all copies of the shared secret Z and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated. After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 6 above).

Party V **shall** execute the following key-agreement transformation to a) establish a shared secret value Z with party U, and b) derive secret keying material from Z.

Actions: V derives secret keying material as follows:

1. Receive an ephemeral public key $Q_{e,U}$ (purportedly) from U. If $Q_{e,U}$ is not received, return an error indicator without performing the remaining actions.
2. Verify that $Q_{e,U}$ is a valid public key for the parameters D as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, return an error indicator without performing the remaining actions.
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret Z_s from the set of domain parameters D, V's static private key $d_{s,V,}$ and U's static public key $Q_{s,U.}$ If the call to the ECC CDH primitive outputs an error indicator,, destroy the results of all intermediate calculations used in the attempted computation of Z_s, and return an error indicator without performing the remaining actions.
4. Use the ECC CDH primitive to derive a shared secret Z_e from the set of domain parameters D, V's static private key $d_{s,V,}$ and U's ephemeral public key $Q_{e,U.}$ If this call to the ECC CDH primitive outputs an error indicator, destroy Z_s,, destroy the results of all intermediate calculations used in the attempted computation of Z_e, and return an error indicator without performing the remaining actions.

5. Compute the shared secret $Z = Z_e \| Z_s$. Destroy Z_e and Z_s.
6. Use the agreed-upon key derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value Z and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of Z, and return an error indicator without performing the remaining action.
7. Destroy all copies of the shared secret Z and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation can be incorporated into this scheme. See Section 6.2.1.5 for details.

The One-Pass Unified Model is summarized in Table 20.

Table 20: One-Pass Unified Model Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	(q, *FR*, a, b{, *SEED*}, G, n, h)	(q, *FR*, a, b{, *SEED*}, G, n, h)
Static Data	Static private key $d_{s,U}$ Static public key $Q_{s,U}$	Static private key $d_{s,V}$ Static public key $Q_{s,V}$
Ephemeral Data	Ephemeral private key $d_{e,U}$ Ephemeral public key $Q_{e,U}$	N/A
Computation	1. Compute Z_s by calling ECC CDH using $d_{s,U}$ and $Q_{s,V}$ 2. Compute Z_e by calling ECC CDH using $d_{e,U}$ and $Q_{s,V}$ 3. Compute $Z = Z_e \| Z_s$	1. Compute Z_s by calling ECC DH using $d_{s,V}$ and $Q_{s,U}$ 2. Compute Z_e by calling ECC DH using $d_{s,V}$ and $Q_{e,U}$ 3. Compute $Z = Z_e \| Z_s$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy Z	1. Compute *DerivedKeyingMaterial* 2. Destroy Z

6.2.1.3 MQV1, C(1e, 2s, FFC MQV) Scheme

This section describes the MQV1 scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.2.1 are true. In particular, it is assumed that party U has obtained the static public key y_V of party V, and party V has obtained the static public key y_U of party U.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following key-agreement transformation in order to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair (r_U, t_U) from the domain parameters *D* as specified in Section 5.6.1.1. Send the public key t_U to V.
2. Use the MQV1 form of the FFC MQV primitive in Section 5.7.2.1.2 to derive a shared secret *Z* from the set of domain parameters *D*, U's static private key x_U, V's static public key y_V, U's ephemeral private key r_U, U's ephemeral public key t_U, and (for a second time) V's static public key y_V. If the call to the FFC MQV primitive outputs an error indicator, destroy the ephemeral private key r_U, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key r_U, and return an error indicator without performing the remaining actions.
4. If the ephemeral private key r_U will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy r_U.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 4 above).

Party V **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party U, and b) derive secret keying material from *Z*.

Actions: V derives secret keying material as follows:

1. Receive an ephemeral public key t_U (purportedly) from U. If t_U is not received, return an error indicator without performing the remaining actions.
2. Verify that t_U is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, return an error indicator without performing the remaining actions.
3. Use the MQV1 form of the FFC MQV primitive in Section 5.7.2.1.2 to derive a shared secret *Z* from the set of domain parameters *D*, V's static private key x_V, U's static public

key y_U, V's static private key x_V (for a second time), V's static public key y_V, and U's ephemeral public key t_U. If the call to the FFC MQV primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.

4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining action.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation can be incorporated into this scheme. See Section 6.1.1.5 for details.

MQV1 is summarized in Table 21.

Table 21: MQV1 Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	(*p, q, g*{, *SEED, pgenCounter*})	(*p, q, g*{, *SEED, pgenCounter*})
Static Data	Static private key x_U Static public key y_U	Static private key x_V Static public key y_V
Ephemeral Data	Ephemeral private key r_U Ephemeral public key t_U	N/A
Computation	Compute *Z* by calling FFC MQV using x_U, y_V, r_U, t_U, and y_V (again)	Compute *Z* by calling FFC MQV using x_V, y_U, x_V (again), y_V, and t_U
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

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

This section describes the One-Pass MQV scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.2.1 are true. In particular, it is assumed that party U has obtained the static public key $Q_{s,V}$ of party V, and party V has obtained the static public key $Q_{s,U}$ of party U.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) from the domain parameters *D* as specified in Section 5.6.1.2. Send the public key $Q_{e,U}$ to V.
2. Use the One-Pass MQV form of the ECC MQV primitive in Section 5.7.2.3.2 to derive a shared secret value *Z* from the set of domain parameters *D*, U's static private key $d_{s,U}$, V's static public key $Q_{s,V}$, U's ephemeral private key $d_{e,U}$, U's ephemeral public key $Q_{e,U}$, and (for a second time) V's static public key $Q_{s,V}$. If the call to the ECC MQV primitive outputs an error indicator, destroy the ephemeral private key $d_{e,U}$, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.
4. If the ephemeral private key $d_{e,U}$ will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy $d_{e,U}$**.**
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 4 above).

Party *V* **shall** execute the following transformation to a) establish a shared secret value *Z* with party U, and b) derive shared secret keying material from *Z*.

Actions: V derives secret keying material as follows:

1. Receive an ephemeral public key $Q_{e,U}$ (purportedly) from U. If $Q_{e,U}$ is not received, return an error indicator without performing the remaining actions.
2. Verify that $Q_{e,U}$ is a valid public key for the parameters *D* as specified in Section 5.6.2.3.2 or 5.6.2.3.3. If assurance of public key validity cannot be obtained, return an error indicator without performing the remaining actions.
3. Use the One-Pass MQV form of the ECC MQV primitive in Section 5.7.2.3.2 to derive a shared secret value *Z* from the set of domain parameters *D*, V's static private key $d_{s,V}$, U's static public key $Q_{s,U}$, V's static private key $d_{s,V}$ (for a second time), V's static public key $Q_{s,V}$, and U's ephemeral public key $Q_{e,U}$. If the call to the ECC MQV primitive outputs an

error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.

4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining action.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation is to be incorporated into this scheme. See Section 6.2.1.5 for details.

The One-Pass MQV scheme is summarized in Table 22.

Table 22: One-Pass MQV Model Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	(*q, FR, a, b*{, *SEED*}, *G, n, h*)	(*q, FR, a, b*{, *SEED*}, *G, n, h*)
Static Data	Static private key $d_{s,U}$ Static public key $Q_{s,U}$	Static private key $d_{s,V}$ Static public key $Q_{s,V}$
Ephemeral Data	Ephemeral private key $d_{e,U}$ Ephemeral public key $Q_{e,U}$	N/A
Computation	Compute *Z* by calling ECC MQV using $d_{s,U}$, $Q_{s,V}$, $d_{e,U}$, $Q_{e,U}$, and $Q_{s,V}$ (again)	Compute *Z* by calling ECC MQV using $d_{s,V}$, $Q_{s,U}$, $d_{s,V}$ (again), $Q_{s,V}$, and $Q_{e,U}$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

6.2.1.5 Incorporating Key Confirmation into a C(1e, 2s) Scheme

The subsections that follow illustrate how to incorporate key confirmation (as described in Section 5.9) into the C(1e, 2s) key-agreement schemes described above. Note that V cannot act as a key-confirmation recipient unless a nonce ($Nonce_V$) is provided by V to U and is used (in addition to the shared secret *Z*) as input to the key-derivation method employed by the scheme. In terms of the preceding descriptions of C(1e, 2s) schemes, this would be accomplished by including $Nonce_V$ (in a protocol-specific fashion) in the *OtherInput* used during key derivation.

The flow depictions separate the key-establishment flow from the key-confirmation flow. The depictions and accompanying discussions presume that the assumptions of the scheme have been satisfied, that the key-agreement transaction has proceeded successfully through key derivation, and that the received *MacTags* are successfully verified as specified in Section 5.2.2.

6.2.1.5.1 C(1e, 2s) Scheme with Unilateral Key Confirmation Provided by U to V

Figure 10 depicts a typical flow for a C(1e, 2s) scheme with unilateral key confirmation from party U to party V. In this situation, party U and party V assume the roles of key-confirmation provider and recipient, respectively. Since V does not contribute an ephemeral public key during the key-agreement process, a nonce ($Nonce_V$) **shall** be provided by V to U and used (in addition to the shared secret *Z*) as input to the key-derivation method employed by the scheme. $Nonce_V$ is used as $EphemData_V$ during *MacTag* computations. The successful completion of the key confirmation process provides party V with assurance that party U has derived the same secret *Z* value. If $Nonce_V$ is a *random nonce*, then party V also obtains assurance that party U has actively participated in the process; see Section 5.4 for a discussion of the length and security strength required for the nonce.

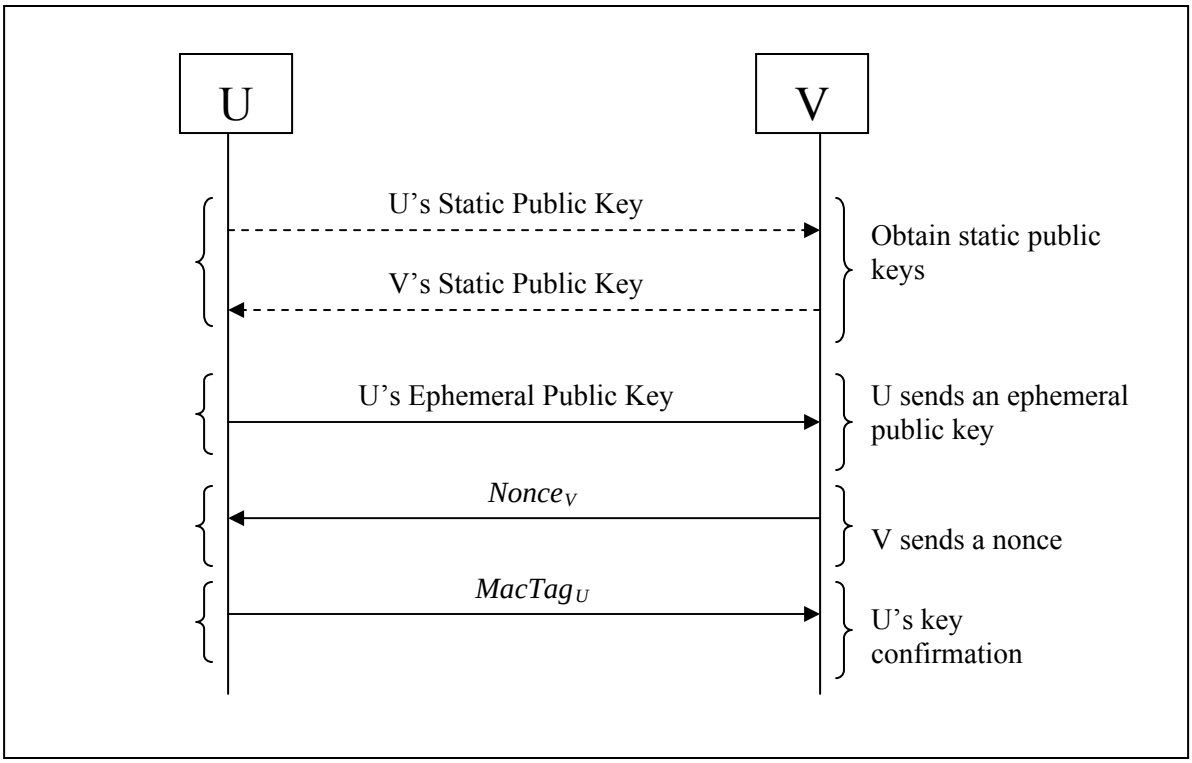

Figure 10: C(1e, 2s) scheme with unilateral key confirmation from U to V

To provide (and receive) key confirmation (as described in Section 5.9.1.1), U (and V) set

$EphemData_U = EphemPubKey_U$, and $EphemData_V = Nonce_V$:

Party U provides $MacTag_U$ to V (as specified in Section 5.9.1.1, with $P = U$ and $R = V$), where $MacTag_U$ is computed (as specified in Section 5.2.1) using

$MacData_U$ = "KC_1_U" || ID_U || ID_V || $EphemPubKey_U$ || $Nonce_V$ {|| $Text_U$}.

Party V (the key-confirmation recipient) uses the same format for $MacData_U$ to compute its own version of $MacTag_U$ and then verifies that the newly computed *MacTag* matches the value provided by U.

6.2.1.5.2 C(1e, 2s) Scheme with Unilateral Key Confirmation Provided by V to U

Figure 11 depicts a typical flow for a C(1e, 2s) scheme with unilateral key confirmation from party V to party U. In this scenario, party V and party U assume the roles of key-confirmation provider and recipient, respectively. The successful completion of the key-confirmation process provides party U with a) assurance that party V has derived the same secret *Z* value; and b) assurance that party V has actively participated in the process.

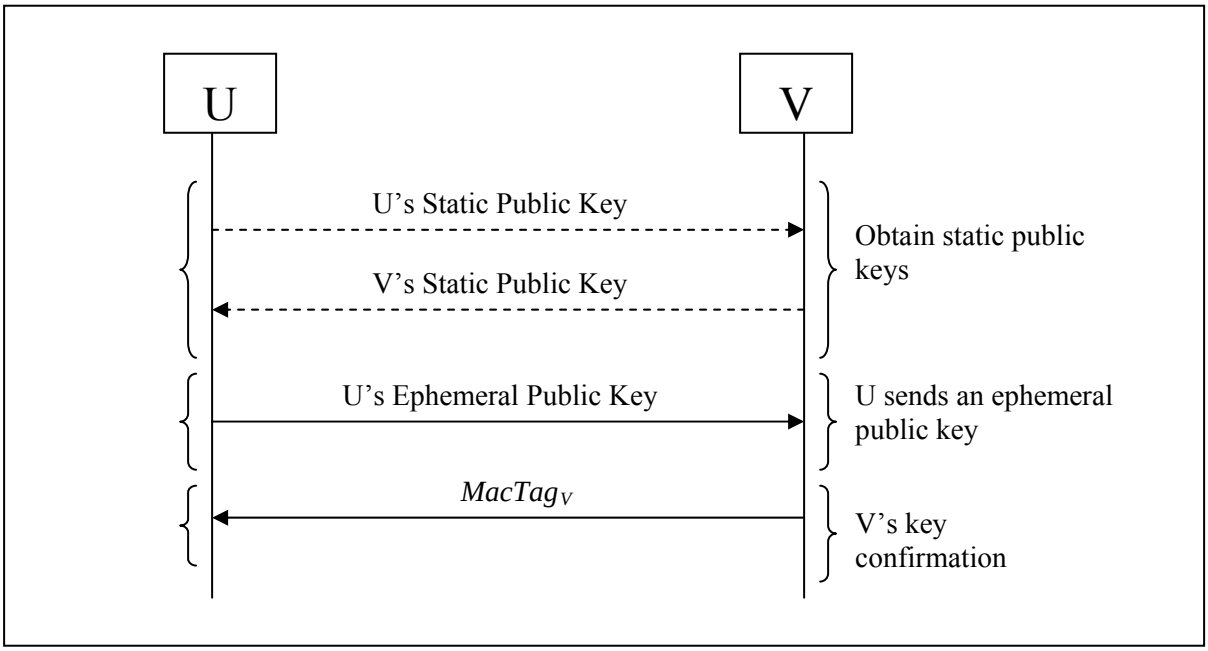

Figure 11: C(1e, 2s) scheme with unilateral key confirmation from V to U

To provide (and receive) key confirmation (as described in Section 5.9.1.1), both parties set

$EphemData_V$ = *Null*, and $EphemData_U$ = $EphemPubKey_U$:

Party V provides $MacTag_V$ to U (as specified in Section 5.9.1.1, with *P* = *V* and *R* = *U*), where $MacTag_V$ is computed (as specified in Section 5.2.1) using

$MacData_V$ = "KC_1_V" || ID_V || ID_U || *Null* || $EphemPubKey_U$ {|| *Text*}.

Party U (the key-confirmation recipient) uses the same format for $MacData_V$ to compute its own version of $MacTag_V$, and then verifies that the newly computed *MacTag* matches the value provided by V.

6.2.1.5.3 C(1e, 2s) Scheme with Bilateral Key Confirmation

Figure 12 depicts a typical flow for a C(1e, 2s) scheme with bilateral key confirmation. In this method, party U and party V assume the roles of both the provider and the recipient in order to obtain bilateral key confirmation. Since V does not contribute an ephemeral public key during the key-agreement process, a nonce ($Nonce_V$) **shall** be provided by V to U and used (in addition to the shared secret *Z*) as input to the key-derivation method employed by the scheme. $Nonce_V$ is used as the $EphemData_V$ during *MacTag* computations. The successful completion of the key-confirmation process provides each party with assurance that the other party has derived the same secret *Z* value. Party U obtains assurance that party V has actively participated in the process; if $Nonce_V$ is a *random nonce*, then party V also obtains assurance that party U has actively participated in the process; see Section 5.4 for a discussion of the length and security strength required for the nonce.

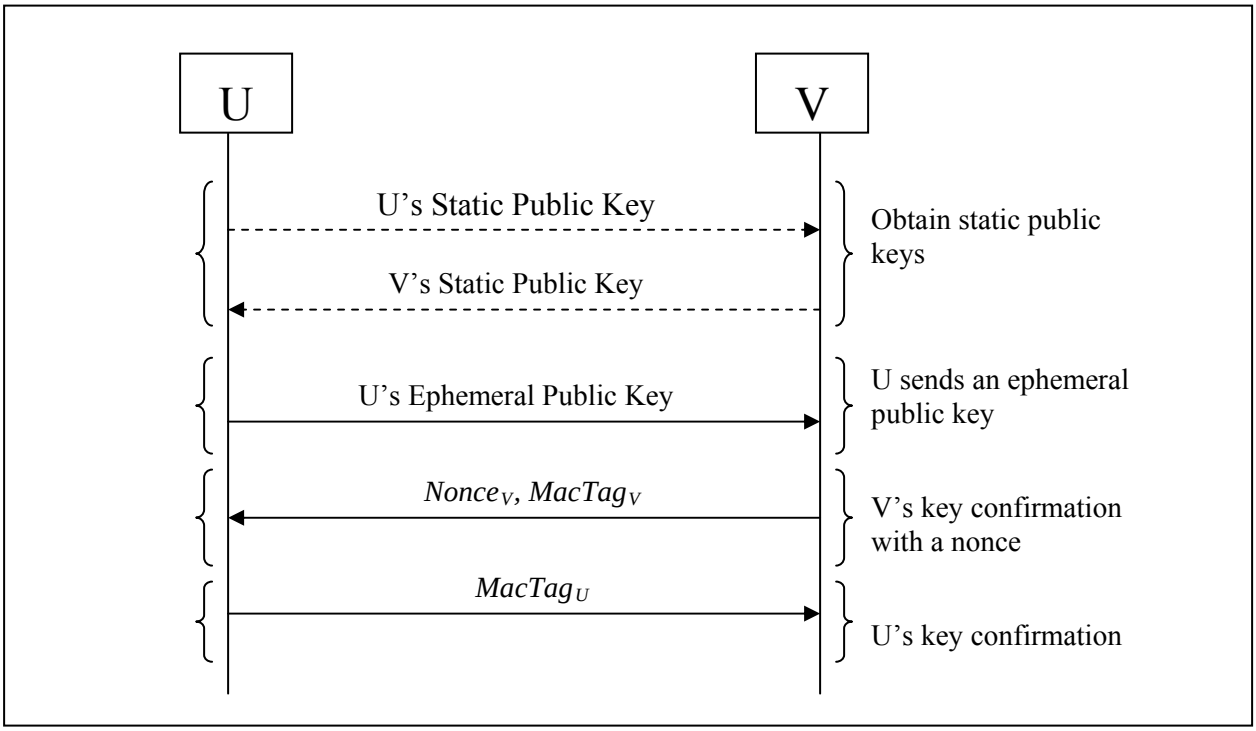

Figure 12: C(1e, 2s) scheme with bilateral key confirmation

To provide bilateral key confirmation (as described in Section 5.9.2.1), U and V exchange and verify *MacTags* that have been computed (as specified in Sections 5.2.1) using

$EphemData_U = EphemPubKey_U$ and $EphemData_V = Nonce_V$:

Party V provides $MacTag_V$ to U (as specified in Sections 5.9.1.1 and 5.9.2.1, with $P = V$ and $R = U$); $MacTag_V$ is computed by V (and verified by U) using

$MacData_V =$ "KC_2_V" $\| ID_V \| ID_U \| Nonce_V \| EphemPubKey_U \{\| Text_V\}$.

Party U provides $MacTag_U$ to V (as specified in Sections 5.9.1.1 and 5.9.2.1, with $P = U$ and $R = V$); $MacTag_U$ is computed by U (and verified by V) using

$$MacData_U = \text{“KC\_2\_U”} \| ID_U \| ID_V \| EphemPubKey_U \| Nonce_V \{\| Text_U\}.$$

Note that in **Error! Reference source not found.**, party V's nonce ($Nonce_V$) and the *MacTag* ($MacTag_V$) are depicted as being sent in the same message (to reduce the number of passes in the combined key-agreement/key-confirmation process). They may also be sent separately (as long as $Nonce_V$ is sent before the *MacTags* are exchanged). The $MacTag_V$ and $MacTag_U$ can be sent in any order, as long as $Nonce_V$ is available to generate and verify both MAC tags.

6.2.2 C(1e, 1s) Schemes

For each of C(1e, 1s) schemes, Party U generates an ephemeral key pair, but uses 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 (for example, from a certificate signed by a trusted CA or directly from party V, who is trusted) 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 secret keying material (see Figure 13).

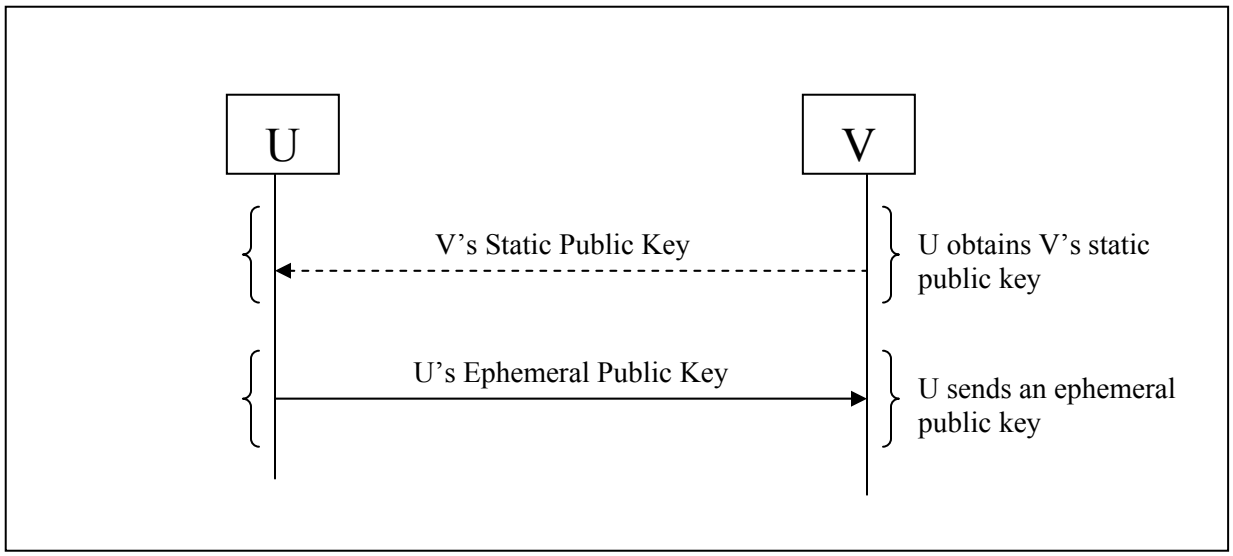

Figure 13: C(1e, 1s) schemes: U contributes an ephemeral key pair, and V contributes a static key pair

Assumptions: In order to execute a C(1e, 1s) key-establishment scheme in compliance with this Recommendation, the following assumptions **shall be** true.

1. Each party has an authentic copy of the same set of domain parameters, *D*. These parameters have been generated as specified in Section 5.5.1. For FFC schemes, $D = (p, q, g\{, SEED, pgenCounter\})$; for ECC schemes, $D = (q, FR, a, b\{, SEED\}, G, n, h)$. Furthermore, each party has obtained assurance of the validity of these domain parameters as specified in Section 5.5.2.

2. Party V has been designated as the owner of a static key pair that was generated as specified in Section 5.6.1 using the set of domain parameters, *D*. For FFC schemes, the static key pair is (x, y); for ECC schemes, the static key pair is (d_s, Q_s). Party V has obtained assurance of the validity of its own static public key as specified in Section

5.6.2.1. Party V has obtained assurance of its possession of the correct value of its own private key as specified in Section 5.6.2.1.5.

3. The parties have agreed upon an **approved** key-derivation method, as well as an **approved** algorithm to be used with that method (e.g., a hash function) and other associated parameters to be used (see Section 5.8).

4. If key confirmation is used, the parties have agreed upon an **approved** MAC and associated parameters (see Table 8 and Table 9 in Section 5.9.3).

5. Prior to or during the key-agreement process, party U receives party V's static public key in a trusted manner (e.g., from a certificate signed by a trusted CA or directly from party V, who is trusted by the recipient) Party U has obtained assurance of the validity of party V's static public key as specified in Section 5.6.2.2.1.

6. When an identifier is used to label either party during the key-agreement process, both parties are aware of the particular identifier employed for that purpose. In particular, when an identifier is used to label party V during the key-agreement process, that identifier has a trusted association to that party V's static public key. (In other words, whenever both the identifier and static public key of one participant are employed in the key-agreement process, they are associated in a manner that is trusted by the other participant.) When an identifier is used to label party U during the key-agreement process, it has been selected/assigned in accordance with the requirements of the protocol relying upon the use of the key-agreement scheme.

The following is an assumption for using the derived keying material for purposes beyond the C(1e,1s) scheme itself.

Party U has obtained assurance that party V is (or was) in possession of the appropriate static private key, as specified in Section 5.6.2.2.3.

6.2.2.1 dhOneFlow, C(1e, 1s, FFC DH) Scheme

This section describes the dhOneFlow scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.2.2 are true. In particular, it is assumed that party U has obtained the static public key y_V of party V.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key-derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair (r_U, t_U) from the domain parameters *D* as specified in Section 5.6.1.1. Send the public key t_U to V.

2. Use the FCC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* from the set of domain parameters *D*, U's ephemeral private key r_U, and V's static public key y_V. If the call to the FFC DH primitive outputs an error indicator, destroy the ephemeral private

key r_U, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.

3. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key r_U, and return an error indicator without performing the remaining actions.
4. If the ephemeral private key r_U will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy r_U.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 4 above).

Party V **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party U, and b) derive secret keying material from *Z*.

Actions: V derives secret keying material as follows:

1. Receive an ephemeral public key t_U (purportedly) from U. If t_U is not received, return an error indicator without performing the remaining actions.
2. Verify that t_U is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, return an error indicator without performing the remaining actions.
3. Use the FCC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* from the set of domain parameters *D*, V's static private key x_V, and U's ephemeral public key t_U. If the call to the FFC DH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining action.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation can be incorporated into this scheme. See Section 6.2.2.3 for details.

dhOneFlow is summarized in Table 23.

Table 23: **dhOneFlow Key-agreement Scheme Summary**

	Party U	**Party V**
Domain Parameters	(*p, q, g*{, *SEED, pgenCounter*})	(*p, q, g*{, *SEED, pgenCounter*})
Static Data	N/A	Static private key x_V Static public key y_V
Ephemeral Data	Ephemeral private key r_U Ephemeral public key t_U	N/A
Computation	Compute *Z* by calling FFC DH using r_U and y_V	Compute *Z* by calling FFC DH using x_V and t_U
Derive Secret Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

6.2.2.2 (Cofactor) One-Pass Diffie-Hellman, C(1e, 1s, ECC CDH) Scheme

This section describes the One-Pass Diffie-Hellman scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.2.2 are true. In particular, it is assumed that party U has obtained the static public key $Q_{s,V}$ of party V.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key derivation function.

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Generate an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) from the domain parameters *D* as specified in Section 5.6.1.2. Send the public key $Q_{e,U}$ to V.
2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* from the set of domain parameters *D*, U's ephemeral private key $d_{e,U}$, and V's static public key $Q_{s,V}$. If this call to the ECC CDH primitive outputs an error indicator, destroy the ephemeral private key $d_{e,U}$, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the

ephemeral private key $d_{e,U}$, and return an error indicator without performing the remaining actions.

4. If the ephemeral private key $d_{e,U}$ will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy $d_{e,U}$.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If the ephemeral key pair is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same ephemeral key pair (r_U, t_U) may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 4 above).

Party V **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party U, and b) derive secret keying material from *Z*.

Actions: V derives secret keying material as follows:

1. Receive an ephemeral public key $Q_{e,U}$ (purportedly) from U. If $Q_{e,U}$ is not received, return an error indicator without performing the remaining actions.
2. Verify that $Q_{e,U}$ is a valid public key for the parameters *D* as specified in Section 5.6.2.3. If assurance of public key validity cannot be obtained, return an error indicator without performing the remaining actions.
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* from the set of domain parameters *D*, V's static private key $d_{s,V,}$ and U's ephemeral public key $Q_{e,U.}$ If this call to the ECC CDH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V, if available). (See Section 5.8.) If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining action.
6. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation can be incorporated into this scheme. See Section 6.2.2.3 for details.

The One-Pass Diffie-Hellman is summarized in Table 24.

Table 24: One-Pass Diffie-Hellman Key-agreement Scheme Summary

	Party U	Party V
Domain Parameters	(*q, FR, a, b*{, *SEED*}, *G, n, h)*	(*q, FR, a, b*{, *SEED*}, *G, n, h)*
Static Data	N/A	Static private key $d_{s,V}$ Static public key $Q_{s,V}$
Ephemeral Data	Ephemeral private key $d_{e,U}$ Ephemeral public key $Q_{e,U}$	N/A
Computation	Compute *Z* by calling ECC CDH using $d_{e,U}$ and $Q_{s,V}$	Compute *Z* by calling ECC CDH using $d_{s,V}$ and $Q_{e,U}$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* 2. Destroy *Z*

6.2.2.3 Incorporating Key Confirmation into a C(1e, 1s) Scheme

The subsection that follows illustrates how to incorporate key confirmation (as described in Section 5.9) into the C(1e, 1s) key-agreement schemes described above. Note that only unilateral key confirmation from V to U is specified, since only party V has a static key pair that is used in the key-establishment process.

The flow depiction separates the key-establishment flow from the key-confirmation flow. The depiction and accompanying discussion presumes that the assumptions of the scheme have been satisfied, that the key-agreement transaction has proceeded successfully through key derivation, and that the received *MacTag* is successfully verified as specified in Section 5.2.2.

6.2.2.3.1 C(1e, 1s) Scheme with Unilateral Key Confirmation Provided by V to U

Figure 14 depicts a typical flow for a C(1e, 1s) scheme with unilateral key confirmation from party V to party U. In this scenario, party V and party U assume the roles of the key-confirmation provider and recipient, respectively. The successful completion of the key-confirmation process provides party U with a) assurance that party V has derived the same secret *Z* value; and b) assurance that party V has actively participated in the process.

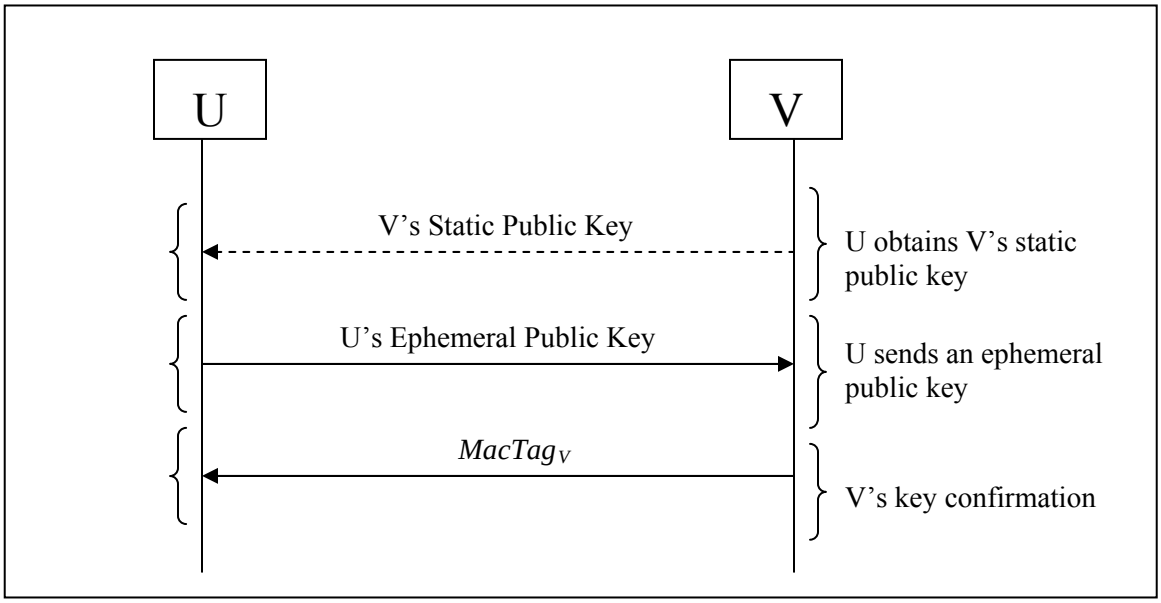

Figure 14: C(1e, 1s) scheme with unilateral key confirmation from V to U

To provide (and receive) key confirmation (as described in Section 5.9.1.1), both parties set

$EphemData_V = Null$, and $EphemData_U = EphemPubKey_U$:

Party V provides $MacTag_V$ to U (as specified in Section 5.9.1.1, with $P = V$ and $R = U$), where $MacTag_V$ is computed (as specified in Section 5.2.1) using

$MacData_V$ = "KC_1_V" || ID_V || ID_U || $Null$ || $EphemPubKey_U$ {|| $Text_V$}.

Party U (the key-confirmation recipient) uses the same format for $MacData_V$ to compute its own version of $MacTag_V$ and then verifies that the newly computed $MacTag$ matches the value provided by V.

6.3 C(0e, 2s) Schemes

In this category, the parties use only static key pairs. Each party obtains the other party's static public keys. A nonce is sent by party U to party V to ensure that the derived keying material is different for each key-establishment transaction. The parties calculate the shared secret using their own static private key and the other party's static public key. Secret keying material is derived using the key-derivation method, the shared secret, and the nonce (see Figure 15).

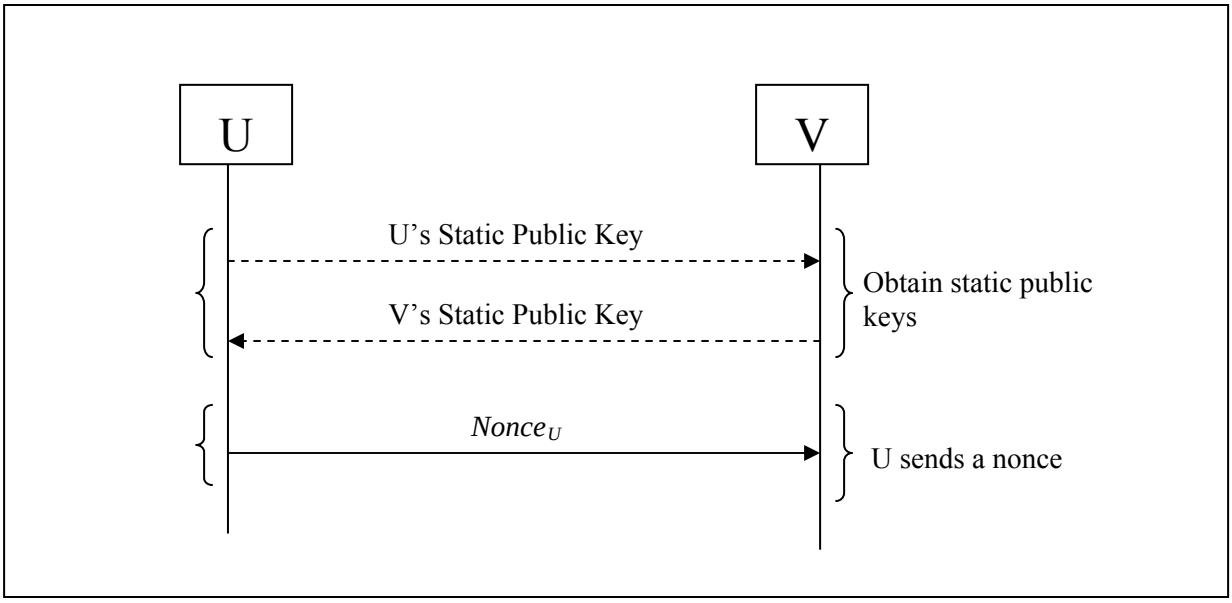

Figure 15: C(0e, 2s) schemes: each party contributes only a static key pair

Assumptions: In order to execute a C(0e, 2s) key-establishment scheme in compliance with this Recommendation, the following assumptions **shall be** true.

1. Each party has an authentic copy of the same set of domain parameters, *D*. These parameters have been generated as specified in Section 5.5.1. For FFC schemes, *D* = (*p, q, g*{, *SEED, pgenCounter*}); for ECC schemes, *D* = (*q, FR, a, b*{, *SEED*}*, G, n, h*). Furthermore, each party has assurance of the validity of these domain parameters as specified in Section 5.5.2.

2. Each party has been designated as the owner of a static key pair that was generated as specified in Section 5.6.1 using the set of Domain parameters, *D*. For FFC schemes, the static key pair is (*x*, *y*); for ECC schemes, the static key pair is (d_s, Q_s). Each party has obtained assurance of the validity of its own static public key as specified in Section 5.6.2.1. Each party has obtained assurance of its possession of the correct value for its own private key as specified in Section 5.6.2.1.5.

3. The parties have agreed upon an **approved** key-derivation method (see Section 5.8), as well as an **approved** algorithm (e.g., a hash function) appropriate for use with the key-derivation method and other associated parameters to be used (see Section 5.8.1). In addition, the parties have agreed on the form of the nonce (see Section 5.4). If key confirmation is used, the parties have agreed upon the form of the **approved** MAC and associated parameters (see Table 8 and Table 9 in Section 5.9.3).

4. Prior to or during the key-agreement process, each party receives the other party's static public key in a trusted manner (e.g., from a certificate signed by a trusted CA or directly from the other party, who is trusted by the recipient). Each party has obtained assurance of the validity of the other party's static public key as specified in Section 5.6.2.2.

5. The recipient of a static public key has obtained assurance that its (claimed) owner is (or was) in possession of the corresponding static private key, as specified in Section 5.6.3.2.

6. When an identifier is used to label a party during the key-agreement process, that identifier has a trusted association to that party's static public key. (In other words, whenever both the identifier and static public key of one participant are employed in the key-agreement process, they are associated in a manner that is trusted by the other participant.) When an identifier is used to label a party during the key-agreement process, both parties are aware of the particular identifier employed for that purpose.

6.3.1 dhStatic, C(0e, 2s, FFC DH) Scheme

This section describes the dhStatic scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.3 are true. In particular, it is assumed that party U has obtained the static public key y_V of party V, and party V has obtained the static public key y_U of party U.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key-derivation function. In particular, this requirement applies to the placement of NonceU in the PartyUInfo subfield of OtherInfo used by the key-derivation method (see Section 5.8).

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Obtain a nonce, $Nonce_U$ (see Section 5.4). Send $Nonce_U$ to V.
2. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* from the set of domain parameters *D*, U's static private key x_U, and V's static public key y_V. If the call to the FFC DH primitive outputs an error indicator, destroy $Nonce_U$, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers IDu and IDv, and $Nonce_U$.). $Nonce_U$ **shall** be in the *PartyUInfo* subfield of *OtherInfo*. If the key-derivation method outputs an error indicator, destroy $Nonce_U$ and all copies of *Z*, and return an error indicator without performing the remaining actions.
4. If $Nonce_U$ will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy $Nonce_U$.
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If $Nonce_U$ is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same $Nonce_U$ may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 4 above).

Party V **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party U, and b) derive secret keying material from *Z*.

Actions: V derives secret keying material as follows:

1. Obtain U's nonce, $Nonce_U$, from U. If $Nonce_U$ is not available, return an error indicator without performing the remaining actions.
2. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret from the set of domain parameters *D*, V's static private key x_V, and U's static public key y_U. If the call to the FFC DH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and ID_V (if available), and $Nonce_U$). $Nonce_U$ **shall** be in the *PartyUInfo* subfield of *OtherInfo*. If the key-derivation method outputs an error indicator, destroy $Nonce_U$ and all copies of *Z*, and return an error indicator without performing the remaining action.
4. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation can be incorporated into this scheme. See Section 6.3.3 for details.

dhStatic is summarized in Table 25.

Table 25: dhStatic Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	(*p, q, g*{, *SEED, pgenCounter*})	(*p, q, g*{, *SEED, pgenCounter*})
Static Data	Static private key x_U Static public key y_U	Static private key x_V Static public key y_V
Ephemeral Data	$Nonce_U$	
Computation	Compute *Z* by calling FFC DH using x_U, and y_V	Compute *Z* by calling FFC DH using x_V, and y_U
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* using $Nonce_U$ 2. Destroy *Z*	1. Compute *DerivedKeyingMaterial* using $Nonce_U$ 2. Destroy *Z*

6.3.2 (Cofactor) Static Unified Model, C(0e, 2s, ECC CDH) Scheme

This section describes the Static Unified Model scheme. Assurance of secure key establishment using this scheme can only be obtained when the assumptions in Section 6.3 are true. In particular, it is assumed that party U has obtained the static public key $Q_{s,V}$ of party V, and party V has obtained the static public key $Q_{s,U}$ of party U.

In this scheme, each party has different actions, which are presented separately below. However, note that U and V must use identical orderings of the bit strings that are input to the key-derivation function. The same requirement applies to the placement of $Nonce_U$ in the *PartyUInfo* subfield of *OtherInfo* (see item 4 below).

Party U **shall** execute the following key-agreement transformation to a) establish a shared secret value *Z* with party V, and b) derive secret keying material from *Z*.

Actions: U generates a shared secret and derives secret keying material as follows:

1. Obtain a nonce, $Nonce_U$ (see Section 5.4). Send $Nonce_U$ to V.
2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* from the set of domain parameters *D*, U's static private key $d_{s,U,}$ *and* V's static public key $Q_{s,V}$**.** If the call to the ECC CDH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of *Z*, and return an error indicator without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers ID_U and $ID_{V,}$ and $Nonce_U$). $Nonce_U$ **shall** be in the *PartyUInfo* subfield of *OtherInfo.* If the key-derivation method outputs an error indicator, destroy all copies of *Z*, and return an error indicator without performing the remaining actions.
4. If $Nonce_U$ will not be used in a broadcast scenario (see Section 7) for subsequent key-establishment transactions using this scheme, then destroy $Nonce_U$
5. Destroy all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: If $Nonce_U$ is used in a broadcast scenario by party U (see Section 7) for subsequent key-establishment transactions using this scheme, then the same $Nonce_U$ may be used in other key-establishment transactions occurring during the same broadcast (i.e., step 1 above would not be repeated). After the final broadcast transaction, the ephemeral private key r_u **shall** be destroyed (see step 4 above).

Party V **shall** execute the following key-agreement transformation to a) establish a shared secret value, *Z,* with party U, and b) derive secret keying material from *Z*.

Actions: V derives secret keying material as follows:

1. Obtain U's nonce, $Nonce_{U,}$ from U. If $Nonce_U$ is not if available, output an error indicator and stop.

2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret Z from the set of domain parameters D, V's static private key $d_{s,V}$, and U's static public key $Q_{s,U}$. If the call to the ECC CDH primitive outputs an error indicator, destroy the results of all intermediate calculations used in the attempted computation of Z, and return an error indicator without performing the remaining actions.
3. Use the agreed-upon key derivation method to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value Z and *OtherInput* (including the identifiers ID_U and ID_V (if available), and $Nonce_U$. $Nonce_U$ **shall** be in the *PartyUInfo* subfield of *OtherInfo.* If the key-derivation method outputs an error indicator, destroy all copies of Z, and return an error indicator without performing the remaining action.
4. Destroy all copies of the shared secret Z and output *DerivedKeyingMaterial*.

Output: The bit string *DerivedKeyingMaterial* of length *keydatalen* bits or an error indicator.

Note: Key confirmation can be incorporated into this scheme. See Section 6.3.3 for details.

Static Unified Model is summarized in Table 26.

Table 26: Static Unified Model Key-agreement Scheme Summary

	Party U	**Party V**
Domain Parameters	(*q, FR, a, b*{, *SEED*}, *G, n, h)*	(*q, FR, a, b*{, *SEED*}, *G, n, h)*
Static Data	Static private key $d_{s,U}$ Static public key $Q_{s,U}$	Static private key $d_{s,V}$ Static public key $Q_{s,V}$
Ephemeral Data	$Nonce_U$	
Computation	Compute Z by calling ECC CDH using $d_{s,U}$, and $Q_{s,V}$	Compute Z by calling ECC CDH using $d_{s,V}$, and $Q_{s,U}$
Derive Secret Keying Material	1. Compute *DerivedKeyingMaterial* using $Nonce_U$ 2. Destroy Z	1. Compute *DerivedKeyingMaterial* using $Nonce_U$ 2. Destroy Z

6.3.3 Incorporating Key Confirmation into a C(0e, 2s) Scheme

The subsections that follow illustrate how to incorporate key confirmation (as described in Section 5.9) into the C(0e, 2s) key-agreement schemes described above. Note that party V cannot act as a key-confirmation recipient unless a nonce ($Nonce_V$) is provided by V to U and is used (in addition to the shared secret Z) as input to the key-derivation method employed by the scheme. In terms of the preceding descriptions of C(0e, 2s) schemes, this would be accomplished by including $Nonce_V$ (in a protocol-specific fashion) in the *OtherInput* used during key derivation.

The flow depictions separate the key-establishment flow from the key-confirmation flow. The depictions and accompanying discussions presume that the assumptions of the scheme have been satisfied, that the key-agreement transaction has proceeded successfully through key derivation, and that the received *MacTags* are successfully verified as specified in Section 5.2.2.

6.3.3.1 C(0e, 2s) Scheme with Unilateral Key Confirmation Provided by U to V

Figure 16 depicts a typical flow for a C(0e, 2s) scheme with unilateral key confirmation from party U to party V. In this scenario, party U and party V assume the roles of key-confirmation provider and recipient, respectively. A nonce ($Nonce_V$) **shall** be provided by V to U and used (in addition to the shared secret Z and the nonce provided by party U) as input to the key-derivation method employed by the scheme. $Nonce_V$ is used as the $EphemData_V$ during *MacTag* computations. The successful completion of the key-confirmation process provides party V with assurance that party U has derived the same secret Z value. If $Nonce_V$ is a *random nonce*, then party V also obtains assurance that party U has actively participated in the process; see Section 5.4 for a discussion of the length and security strength required for the nonce.

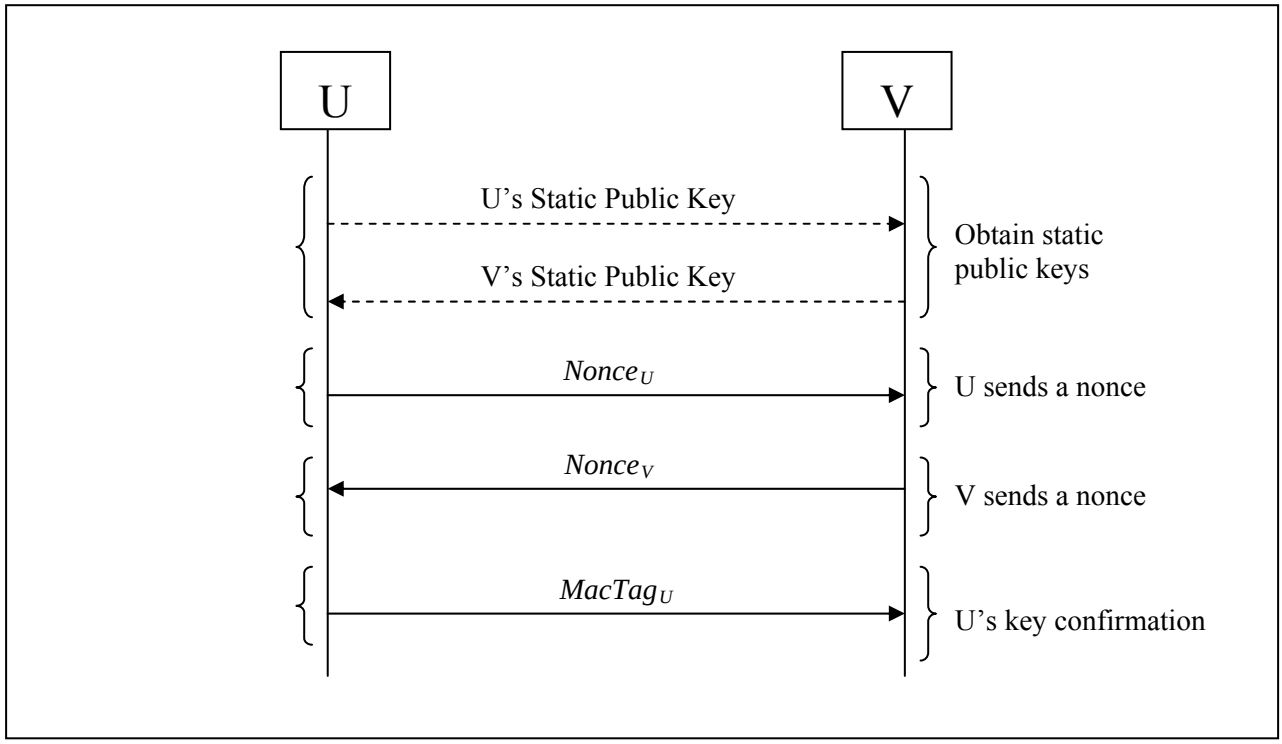

Figure 16: C(0e, 2s) scheme with unilateral key confirmation from U to V

To provide (and receive) key confirmation (as described in Section 5.9.1.1), U (and V) set

$EphemData_U = Nonce_U$, and $EphemData_V = Nonce_V$:

Party U provides $MacTag_U$ to V (as specified in Section 5.9.1.1, with $P = U$ and $R = V$), where $MacTag_U$ is computed (as specified in Section 5.2.1) using

$MacData_U$ = "KC_1_U" || ID_U || ID_V || $Nonce_U$ || $Nonce_V$ {|| $Text_U$}.

Party V (the key-confirmation recipient) uses the same format for $MacData_U$ to compute its own version of $MacTag_U$ and then verifies that the newly computed *MacTag* matches the value provided by U.

6.3.3.2 C(0e, 2s) Scheme with Unilateral Key Confirmation Provided by V to U

Figure 17 depicts a typical flow for a C(0e, 2s) scheme with unilateral key confirmation from party V to party U. In this situation, party V and party U assume the roles of key-confirmation provider and recipient, respectively. The successful completion of the key-confirmation process provides party U with assurance that party V has derived the same secret *Z* value; if $Nonce_U$ is a *random nonce*, then party U also obtains assurance that party V has actively participated in the process; see Section 5.4 for a discussion of the length and security strength required for the nonce.

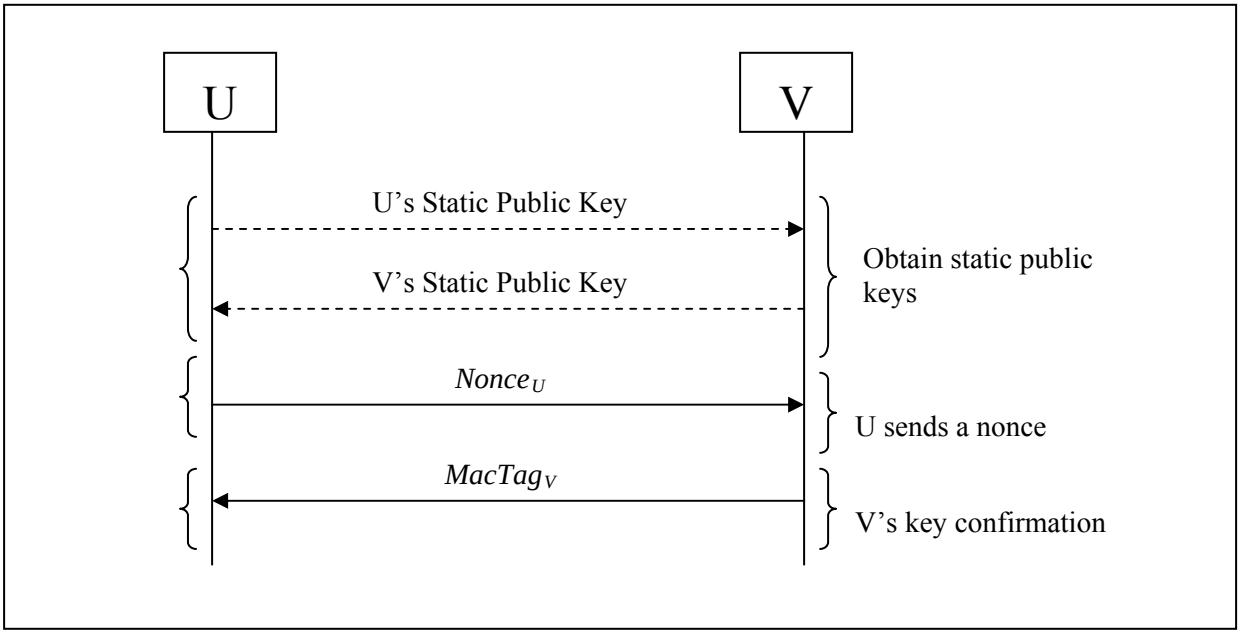

Figure 17: C(0e, 2s) scheme with unilateral key confirmation from V to U

To provide (and receive) key confirmation (as described in Section 5.9.1.1), Both parties set

$EphemData_V = Null$, and $EphemData_U = Nonce_U$:

Party V provides $MacTag_V$ to U (as specified in 5.9.1.1, with $P = V$ and $R = U$), where $MacTag_V$ is computed (as specified in Section 5.2.1) using

$MacData_V$ = "KC_1_V" || ID_V || ID_U || $Null$ || $Nonce_U$ {|| $Text_V$}.

Party U (the key-confirmation recipient) uses the same format for $MacData_V$ to compute its own version of $MacTag_V$, and then verifies that the newly computed *MacTag* matches the value provided by V.

6.3.3.3 C(0e, 2s) Scheme with Bilateral Key Confirmation

Error! Reference source not found. depicts a typical flow for a C(0e, 2s) scheme with bilateral key confirmation. In this method, party U and party V assume the roles of both the provider and the recipient in order to obtain bilateral key confirmation. A nonce ($Nonce_V$) **shall** be provided by V to U and used (in addition to the shared secret *Z* and the nonce, $Nonce_U$, provided by party U) as input to the key-derivation method employed by the scheme. $Nonce_V$ is used as the $EphemData_V$ during *MacTag* computations. The successful completion of the key confirmation process provides each party with assurance that the other party has derived the same secret *Z* value. If $Nonce_U$ is a *random nonce*, then party U obtains assurance that party V has actively participated in the process; if $Nonce_V$ is a *random nonce*, then party V obtains assurance that party U has actively participated in the process. See Section 5.4 for a discussion about the length and security strength required for the nonce.

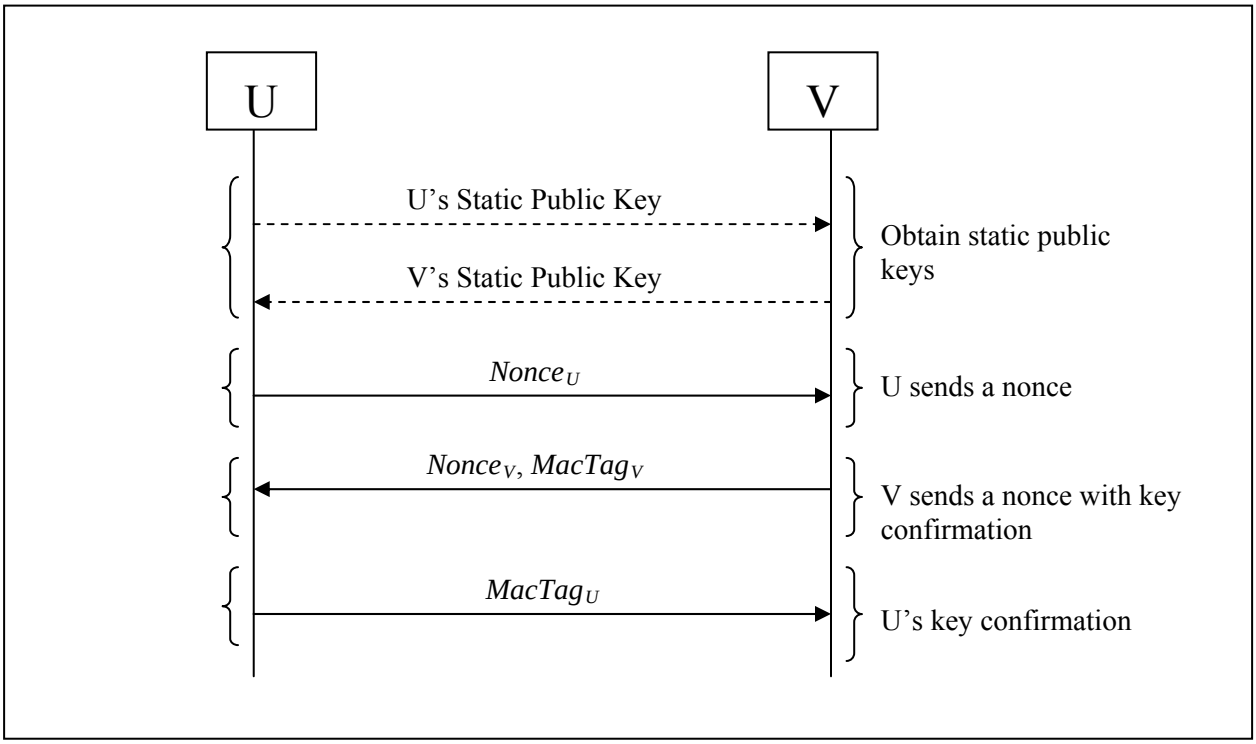

Figure 18: C(0e, 2s) scheme with bilateral key confirmation

To provide bilateral key confirmation (as described in Section 5.9.2.1), U and V exchange and verify *MacTags* that have been computed (as specified in Sections 5.2.1) using

$EphemData_U = Nonce_U$, and $EphemData_V = Nonce_V$:

Party V provides $MacTag_V$ to U (as specified in Sections 5.9.1.1 and 5.9.2.1, with $P = V$ and $R = U$); $MacTag_V$ is computed by V (and verified by U) using

$MacData_V$ = "KC_2_V" || ID_V || ID_U || $Nonce_V$ || $Nonce_U$ {|| $Text_V$}.

Party U provides $MacTag_U$ to V (as specified in Sections 5.9.1.1 and 5.9.2.1, with $P = U$ and $R = V$); $MacTag_U$ is computed by U (and verified by V) using

$MacData_U$ = "KC_2_U" || ID_U || ID_V || $Nonce_U$ || $Nonce_V$ {|| $Text_U$}.

Note that in **Error! Reference source not found.**, party V's nonce ($Nonce_V$) and the *MacTag* ($MacTag_V$) are depicted as being sent in the same message (to reduce the number of passes in the combined key-agreement/key-confirmation process). In fact, they can be sent in other orders and combinations (as long as $Nonce_U$ and $Nonce_V$ are available to generate and verify both MAC tags).

7. DLC-Based Key Transport

A DLC-based key-transport scheme uses both a key-agreement scheme and a key-wrapping algorithm in a single transaction to establish keying material. During this transaction, a key-wrapping key **shall** be established using an **approved** key-agreement scheme. This key **shall** be used by party U to wrap secret keying material using an **approved** key-wrapping algorithm; the wrapped keying material is then sent to party V (i.e., party U in the key-agreement scheme will be the key-transport sender, and party V will be the key-transport receiver). **Approved** key-wrapping algorithms are provided in SP 800-38F.

To comply with this Recommendation, the key-transport transaction **shall** use only **approved** key-agreement schemes that employ party V's static key pair[8] and require an ephemeral contribution by party U[9]. In particular, a C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) key-agreement scheme **shall** be used in which party U is the intended key-transport sender; a C(2e, 0s) scheme **shall not** be used to establish the key-wrapping key (regardless of which party is the intended key-transport sender). Although other methods may be used by protocols that incorporate key transport, this Recommendation makes no statement as to the adequacy of those methods.

Key confirmation may optionally be provided by party V following the unwrapping of the received keying material, either instead of or in addition to any key confirmation that may be performed as part of the key-agreement scheme. When key confirmation is performed following the unwrapping process in accordance with this Recommendation, parties U and V **shall** have agreed upon an **approved** MAC algorithm and associated parameters (see Table 8 and Figure 9 in Section 5.9.3).

7.1 Key Transport Scheme

The DLC-based key-transport scheme is as follows:

1. An agreed-upon C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) key-agreement scheme is used between party U and party V to establish *DerivedKeyingMaterial*, which includes a *KeyWrappingKey* that will subsequently be used by party U for key-transport. Key

[8] To prevent receiver identifier spoofing; since the receiver has used a static key pair during key agreement, the sender has assurance of the identifier of the intended receiver.

[9] To provide the key-transport sender with assurance of the freshness of the key-wrapping key.

confirmation (as specified in Section 5.9 and Section 6) may optionally be incorporated in the key-agreement scheme to provide assurance that the *DerivedKeyingMaterial* is the same for both parties.

2. Party U obtains the *KeyWrappingKey* from the *DerivedKeyingMaterial*.
3. Party U selects secret keying material, *TransportedKeyingMaterial*, to transport to the receiver. If key confirmation is to be performed following key-transport, this *TransportedKeyingMaterial* **shall** include a fresh (i.e., new) *MacKey* to be used for key confirmation and the *KeyData* to be used subsequent to key transport (see Section 7.2).
4. Party U calculates *WrappedKeyingMaterial* = KeyWrap(*KeyWrappingKey, TransportedKeyingMaterial*) using KeyWrap( ), an **approved** key wrapping algorithm.
5. Party U sends *WrappedKeyingMaterial* to party V.
6. Party V receives *WrappedKeyingMaterial* from party U.
7. Party V obtains the *KeyWrappingKey* from the *DerivedKeyingMaterial*.
8. Party V calculates *TransportedKeyingMaterial* = KeyUnwrap(*KeyWrappingKey, WrappedKeyingMaterial*) using KeyUnwrap( ), the key-unwrapping algorithm that corresponds to KeyWrap( ).
9. If key confirmation is to be performed subsequent to key transport to provide assurance to party U that the correct *TransportedKeyingMaterial* has been obtained by party V, then both parties U and V **shall** proceed as specified in Section 7.2.

Note that if the key-agreement scheme used in Step 1 is such that party V does not contribute an ephemeral key pair to the calculation of the shared secret (that is, a C(1e, 2s), C(1e, 1s), or C(0e, 2s) scheme has been used) and key confirmation is not included in the key-agreement scheme, then Steps 1 through 5 can be performed by party U without direct involvement of party V. This can be useful in a store-and-forward environment, such as e-mail.

Key transport schemes can be used in broadcast scenarios. In a broadcast scenario, an exception is made to the rule in this Recommendation that ephemeral keys **shall not** be reused (see Section 5.6.3.3). That is, party U may use the same ephemeral key pair in step 1 above in multiple instances of DLC-based key-agreement (employing the same scheme) if the same secret keying material is being transported to multiple entities for use following key transport[10], and if all these instances of key transport occur "simultaneously" (or within a short period of time). However, the security properties of the key-establishment scheme may be affected by reusing the ephemeral key in this manner.

[10] Note that when key confirmation is performed after key transport, the *MacKey* is different for each instance of key confirmation, but *KeyData* is the same for each key-transport receiver participating in the broadcast (see Section 7.2).

7.2 Key Confirmation for Transported Keying Material

If key confirmation is to be provided in compliance with this Recommendation following the transport of keying material (as specified in Section 7.1), party U **shall** generate a fresh *MacKey* and include it as part of the *TransportedKeyingMaterial* to be transported (see Section 7.1). The transported *MacKey* **shall** be used for the computation and verification of the *MacTag* provided by party V to party U.

For each instance of key confirmation following key transport, this *MacKey* **shall** be generated anew using an **approved** random bit generator that is instantiated at or above the security strength required for the key establishment transaction. In broadcast scenarios, a different *MacKey* **shall** be included in the *TransportedKeyingMaterial* for each key-transport receiver that is expected to provide key confirmation to party U.

The domain parameter set used by the key-agreement scheme employed to establish the key-wrapping key **shall** be used to determine the minimum *MacKey* length and the length of the *MacTag*, as specified in Tables 8 and 9 in Section 5.9.3.

The transported keying material **shall** be formatted as follows:

$$TranportedKeyingMaterial = MacKey \,||\, KeyData.$$

The *KeyData* may be null, or may contain keying material to be used subsequent to key transport. The *MacKey* **shall** be used during key confirmation and then immediately destroyed by both party U and party V.

The *MacKey* portion of the transported keying material and an **approved** MAC algorithm (see Sections 5.2 and 5.9.3) are used by each party to compute a *MacTag* (of an appropriate, agreed-upon length) on the *MacData* (see Section 5.9.1.1) represented as

$$MacData = \text{“KC\_KT”} \,||\, ID_V \,||\, ID_U \,||\, EphemData_V \,||\, EphemData_U \,||\, WrappedKeyingMaterial \,\{\,||\, Text\},$$

where ID_V is the identifier associated with party V, and ID_U is the identifier associated with party U. These identifiers **shall** be the same as those used to label parties U and V during the key-agreement portion of the key-transport transaction. $EphemData_V$ is the ephemeral public key or nonce contributed by party V during the establishment of the key-wrapping key used for key transport; if no ephemeral data was contributed by party V, then *Null* **shall** be used. $EphemData_U$ is the ephemeral public key or nonce that was contributed by party U during the establishment of the key-wrapping key. *WrappedKeyingMaterial* is the ciphertext of the keying material that has been transported, and *Text* is an optional bit string that may be used during key-confirmation that is known by both parties.

Party V (the *MacTag* sender) computes a *MacTag* (using the *MacKey* obtained from the *TranportedKeyingMaterial* and *MacData* formed as described above) and provides it to Party U. Party U (the *MacTag* receiver) computes a *MacTag* (using the *MacKey* that was included in the *TranportedKeyingMaterial* and the *MacData* formed as described above). Party U then verifies that this newly computed *MacTag* matches the *MacTag* value provided by party V

8. Rationale for Selecting a Specific Scheme

The subsections that follow offer possible justifications for selecting schemes from each subcategory, C(*i*e, *j*s). These rationales will provide the user or developer with additional information that may help when making a choice as to which key-agreement scheme to use. The rationales include brief discussions of security properties. They do not present an in-depth discussion of all possible security properties of all schemes. The specific security properties that can be achieved depend on whether a static key is used, whether an ephemeral key is used, how the shared secret is calculated, and whether key confirmation can be incorporated. In general, the security properties for each scheme within a subcategory are the same; when this is not the case, the exceptions are identified.

A scheme should not be chosen based solely on the number of security properties it possesses. Rather, a scheme should be selected based on how well the scheme fulfills system requirements. For instance, if messages are exchanged over a large-scale network where each round trip consumes a considerable amount of time, a scheme with fewer passes might be preferable to a scheme with more passes, even though the latter may possess more security properties. It is important to keep in mind that a key-agreement scheme may be a component of a larger protocol that offers additional security properties beyond those provided by the key-agreement scheme alone. Protocols, per se, are not specified in this Recommendation.

Note: In order to provide concise descriptions of the security properties possessed by the various schemes, it is necessary to make some assumptions concerning the format and type of data that is used during key derivation. These assumptions are made solely for the purposes of Sections 8.1 through 8.6; they are not intended to preclude the options specified elsewhere in this Recommendation. When discussing the rationale for selecting a scheme, it is assumed that the *OtherInfo* input to the key-derivation method employed during a particular key-agreement transaction uses either the concatenation format or the ASN.1 format (see Section 5.8.1.2). In cases where an **approved** extraction-then-expansion key-derivation procedure is employed, it is assumed that this *OtherInfo* is used as the *Context* input during the key-expansion step, as specified in [SP 800-56C]. It is also assumed that *OtherInfo* does not include pre-shared secrets, but that *OtherInfo* does include (at least) sufficiently specific identifiers for the participants in the transaction and an identifier for the key-agreement scheme being used during the transaction. Finally, it is assumed that all required nonces employed during the transaction are random nonces that contain a component consisting of a random bit string formed in accordance with the recommendations of Section 5.4.

8.1 Rationale for Choosing a C(2e, 2s) Scheme

Since these schemes require each participant to own a static key pair that is used in their key-agreement transaction, each party has assurance that no unintended party (i.e., no parties other than the owners of the corresponding static key pairs) can compute the shared secret, *Z*, without the compromise of a static private key.

Since these schemes also require each participant to generate an ephemeral key pair that is used in their transaction, each party has assurance that the shared secret varies from one C(2e, 2s) transaction to the next. Even if both the static and ephemeral private keys of one party from one

transaction are compromised, the shared secrets from other legitimate C(2e, 2s) transactions (i.e., transactions between honest parties) are protected by the use of different ephemeral private keys.

If a particular entity's static private key is acquired by an adversary, then the adversary could masquerade as that entity while engaging in any C(2e, 2s) key-agreement transaction that permits the use of the compromised key pair. If an MQV scheme (MQV2 or MQV) is employed during the transaction, the adversary is limited to using the compromised static private key for that type of masquerade. The use of the Full Unified model or dhHybrid1 scheme, however, offers the adversary additional opportunities for masquerading: If an adversary compromises an entity's static private key, then the adversary may be able to impersonate other parties during a Full Unified model- or dhHybrid1-based key-agreement transaction with that entity. Also, the compromise of the static component of a shared secret formed by two parties using the Full Unified Model or dhHybrid1 scheme will permit an adversary to masquerade as either party to the other party in future key-agreement transactions that rely on the same scheme and the same two static key pairs.

Key confirmation can be provided in either or both directions as part of these schemes by using the methods specified in Section 6.1.1.5. This allows the key-confirmation recipient to obtain assurance that the key-confirmation provider has possession of the *MacKey* derived from the shared secret *Z* and has used it with the appropriate *MacData* to compute the received *MacTag*. In the absence of a compromise of private information[11], a key-confirmation recipient can obtain assurance that the correct identifier is associated with the key-confirmation provider and that the provider is the owner of the static public key associated with that identifier. A key-confirmation recipient can also receive assurance of active (and successful) participation by the key-confirmation provider in the key-agreement transaction.

8.2 Rationale for Choosing a C(2e, 0s) Scheme

These schemes require each participant to generate an ephemeral key pair that is used in their key-agreement transaction. No static key pairs are employed. Because the ephemeral private keys used in the computation of their shared secret are destroyed immediately after use, these schemes offer assurance to each party that the shared secret, *Z*, computed during a legitimate C(2e, 0s) transaction (i.e., one involving two honest parties) is protected against the compromise of shared secrets and/or private keys from other (prior or future) transactions.

Unlike a static public key, which is assumed to have a trusted association with an identifier for its owner, there is no assumption of a trusted association between an ephemeral public key and an identifier. Thus, these schemes offer no assurance to either party of the accuracy of any identifier that may be used to label the entity with whom they have established a shared secret. The use of C(2e, 0s) schemes may be appropriate in applications where any trusted association desired/required between an identifier and an ephemeral public key is enforced by methods external to the scheme.

This Recommendation does not specify how to incorporate key confirmation in these schemes.

[11] For example, a static private key or a static *Z* component.

8.3 Rationale for Choosing a C(1e, 2s) Scheme

These schemes require each participant to own a static key pair that is used in their key-agreement transaction; in addition, party U is required to generate and use an ephemeral key pair. Different assurances are provided to parties U and V by the utilization of a C(1e, 2s) scheme.

The use of static key pairs in the key-agreement transaction provides each party with assurance that no unintended party (i.e., no parties other than the owners of the corresponding static key pairs) can compute the shared secret, *Z*, without the compromise of a static private key.

Party U, whose ephemeral key pair is used in the computations, has assurance that the shared secret will vary from one C(1e, 2s) transaction to the next. Party V cannot obtain such assurance, in general, since party V's contribution to the computation of *Z* is static. Party V can, however, obtain assurance that the derived keying material will vary if, for example, party V contributes a nonce that is used as input to the key-derivation method employed during these transactions (as is required when party V is a recipient of key confirmation performed as specified in this Recommendation).

The compromise of party U's static private key does not, by itself, compromise the shared secret computed during any legitimate C(1e, 2s) transaction (i.e., one involving two honest parties). Likewise, the compromise of only party U's ephemeral private key would not compromise the shared secret *Z* for that transaction. However, the compromise of an entity's static private key may lead to the compromise of the shared secrets computed during past, current, and future C(1e, 2s) transactions in which that entity acts as party V; to compromise the shared secrets, the adversary must also acquire the public keys contributed by whomever acts as party U in those transactions.

If an adversary learns a particular entity's static private key, then, in addition to masquerading as that entity, the adversary could masquerade as anyone else when acting as party U in a C(1e, 2s) transaction in which the compromised entity acts as party V. Similarly, the compromise of the static component of a shared secret formed by two entities using the One-Pass Unified Model or dhHybrid1OneFlow scheme will permit an adversary to masquerade as either entity (while acting as party U) to the other entity (acting as party V) in future key-agreement transactions that rely on the same scheme and the same two static key pairs.

Key confirmation can be provided in either or both directions as part of these schemes by using the methods specified in Section 6.2.1.5. This allows the key-confirmation recipient to obtain assurance that the key-confirmation provider has possession of the *MacKey* derived from the shared secret *Z* and has used it with the appropriate *MacData* to compute the received *MacTag*. In the absence of a compromise of private information[12], a key-confirmation recipient can obtain assurance that the correct identifier has been used to label the key-confirmation provider and that the provider is the owner of the static public key associated with that identifier. A key-

[12] For example, a static private key or a static *Z* component.

confirmation recipient can also receive assurance of active (and successful) participation by the key-confirmation provider in the key-agreement transaction.

8.4 Rationale for Choosing a C(1e, 1s) Scheme

In these schemes, party U is required to generate and use an ephemeral key pair, while party V is required to own a static key pair that is used in the key-agreement transaction. Different assurances are provided to parties U and V by the utilization of a C(1e, 1s) scheme.

The use of party V's static public key provides party U with assurance that no unintended party (i.e., no parties other than party U and the owner of V's static public key) can compute the shared secret, *Z*, without the compromise of private material (e.g., a private key). Party V has no such assurance, in general, since party V has no assurance concerning the accuracy of any identifier that may be used to label party U (unless the protocol using this scheme includes additional elements that establish a trusted association between an identifier for party U and the ephemeral public key that U contributes).

Since party U generates the ephemeral key pair that is used in the computation of *Z*, party U has assurance that the shared secret will vary from one C(1e, 1s) transaction to the next. Party V has no such assurance, since party V's contribution to the computation of *Z* is static.

There is no assurance provided to either party that the security of the shared secret is protected against the compromise of a private key. A compromise of party U's ephemeral private key compromises the shared secret for the particular transaction in which it was used. However, the compromise of party V's static private key may lead to the compromise of shared secrets resulting from past, current, and future C(1e, 1s) transactions in which that entity acts as party V; to compromise the shared secrets, the adversary must also acquire the ephemeral public keys contributed by whomever acts as party U in those transactions.

Party V may provide key confirmation to party U as specified in Section 6.2.2.3. This allows party U to obtain assurance that party V has possession of the *MacKey* derived from the shared secret *Z* and has used it with the appropriate *MacData* to compute the received *MacTag*. In the absence of a compromise of private information (e.g., a private key), party U can obtain assurance that the correct identifier has been used to label party V, and that party V is indeed the owner of the static public key associated with that identifier. Assuming that party V's static private key has not been compromised, the key-confirmation recipient, party U, can also receive assurance of active (and successful) participation by party V in the key-agreement transaction.

8.5 Rationale for Choosing a C(0e, 2s) Scheme

These schemes require each participant to own a static key pair that is used in their key-agreement transaction; in addition, party U is required to generate a nonce, which is sent to party V and used (by both parties) as input to their chosen key-derivation method.

The use of static key pairs in their key-agreement transaction provides each party with assurance that no unintended party (i.e., no parties other than the owners of the corresponding static key pairs) can compute their shared secret, *Z*, without the compromise of a static private key.

Although the value of *Z* is static (as long as the two parties employ the same static key pairs), party U, whose (required) nonce is used in the key-derivation computations, has assurance that the derived keying material will vary from one C(0e, 2s) transaction to the next. In general, party V has no such assurance – unless, for example, party V also contributes a nonce that is used as input to the key-derivation method employed during the transaction (as is required when party V is a recipient of key confirmation performed as specified in this Recommendation).

If the (static) value of *Z* formed by the two entities is ever compromised, then all of the keying material derived in past, current, and future C(0e, 2s) key-agreement transactions between these same two entities that employ these same static key pairs may be compromised as well, since the same *Z* value is used to derive keying material in each instance. However, to compromise the keying material from a particular transaction, the adversary must also acquire the nonce contributed by the entity that acted as party U in that transaction. The compromise of the static *Z* value will also permit an adversary to masquerade as either entity to the other entity in future C(0e, 2s) key-agreement transactions.

If an entity's static private key is compromised, then shared secrets of current, prior and future C(0e, 2s) transactions involving that entity's static key pair may be compromised, irrespective of the role (whether party U or party V) played by the compromised entity. Regardless of what entity acts in the other role, the adversary may be able to compute the shared secret *Z* and proceed to compromise the derived keying material, as described above. To complete the attack against a particular transaction, the adversary must acquire the static public key contributed by the other entity participating in that transaction, as well as the nonce contributed by whichever entity acted as party U during the transaction.

Of course, if a static private key has been compromised by an adversary, then the adversary can masquerade as the owner of the corresponding static key pair in key-agreement transactions with any other party. In addition, the adversary can masquerade as any entity (whether acting as party U or party V) while engaging in a C(0e, 2s) key-agreement transaction with the owner of the compromised key pair.

Key confirmation can be provided in either or both directions as part of these schemes by using the methods specified in Section 6.3.3.1. This allows the key-confirmation recipient to obtain assurance that the key-confirmation provider has possession of the *MacKey* derived from the shared secret *Z* and has used it with the appropriate *MacData* to compute the received *MacTag*. In the absence of a compromise of private information[13], a key-confirmation recipient can obtain assurance that the correct identifier has been used to label the key-confirmation provider, and that the provider is the owner of the static public key associated with that identifier. A key-confirmation recipient can also receive assurance of active (and successful) participation by the key-confirmation provider in the key-agreement transaction.

8.6 Choosing a Key-Agreement Scheme for use in Key Transport

The key-agreement scheme employed while performing DLC-based key transport as specified in this Recommendation is required to be a C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) scheme in

[13] For example, a static private key or a static *Z* value.

which the intended key-transport sender acts as party U, and the intended key-transport receiver acts as party V. The security properties of these schemes (in particular, the assurances that they can provide to parties U and V) have been discussed in the previous sections. In this section, the emphasis is on how the security properties of these schemes affect the assurances that can be provided by key transport.

Note: Unless it is explicitly stated otherwise, the analysis that follows is restricted to key-transport transactions that involve only two parties – the sender (party U) and one receiver (party V). The broadcast scenario (involving multiple receivers) will be addressed briefly at the conclusion of the section.)

The use of party V's static public key provides party U with assurance that no unintended party (i.e., no parties other than party U and the owner of V's static public key) can compute the shared secret *Z*, from which the key-wrapping key is derived, without the compromise of private material (e.g., the private component of one or more key pairs, or a static *Z* value).

When a C(2e, 2s), C(1e, 2s), or C(1e, 1s) scheme is employed, party U generates an ephemeral key pair that is used in the computation of *Z*. This provides assurance to party U that the shared secret and key-wrapping key will vary from one key-transport transaction to the next. Assurance of the freshness of the key-wrapping key can also be obtained by party U when a C(0e, 2s) scheme is employed, since U is required to contribute a nonce (see Section 5.4) that is used in the derivation of the key-wrapping key. In the absence of a compromise of private material[14], the freshness of the key-wrapping key provides assurance to party U (the key-transport sender) that the wrapped keying material transported during that particular transaction is protected against the compromise of key-wrapping keys from other transactions (past or future).

Assuming that no key pairs and/or static *Z* values are compromised, the required use of party V's static public key, together with the required ephemeral contribution from party U provides assurance to party U that only the intended key-transport receiver will be able to derive the (fresh) key-wrapping key and use it to unwrap the transported keying material.

If a C(2e, 2s), C(1e, 2s), or C(0e, 2s) scheme is employed, the use of party U's static public key provides party V (the key-transport receiver) with assurance that no unintended party (i.e., no parties other than party V and the owner of party U's static public key) can compute the shared secret *Z*, from which the key-wrapping key is derived, without the compromise of private material (e.g., the private component of one or more key pairs, or a static *Z* value). The trusted association of an identifier with party U's static public key provides party V with a method for accurately labeling the (purported) key-transport sender.

If a C(1e, 1s) scheme is employed, party U (the key-transport sender) is only required to provide an ephemeral public key to party V. Since there is no assumption of a trusted association between an ephemeral public key and an identifier, the use of a C(1e, 2s) scheme (in and of itself) offers no assurance to party V of the accuracy of any identifier that may be associated with party U. Any trusted association desired/required between an identifier and the (purported)

[14] For example, a static private key or a static *Z* value.

sender (party U) would have to be provided by methods external to the key-establishment scheme.

When a C(2e, 2s) scheme is employed, party V generates an ephemeral key pair that is used in the computation of *Z*. This provides assurance to party V that both the shared secret and the key-wrapping key derived from it will vary from one key-transport transaction to the next. Assurance of the freshness of the key-wrapping key may also be obtained by party V when a C(1e, 2s), C(1e, 1s) or C(0e, 2s) scheme is employed, if party V contributes a nonce (see Section 5.4) that is used in the derivation of the key-wrapping key. In the absence of a compromise of private material, assurance of the freshness of the key-wrapping key would provide assurance to party V that the wrapped keying material transported during that particular transaction is protected against the compromise of key-wrapping keys used in other transactions (past or future).

Key confirmation from party V (the intended key-transport receiver) to party U (the intended key-transport sender) can be incorporated in a C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) scheme (as specified in Section 6.1.1.5.2, Section 6.2.1.5.2, Section 6.2.2.3, or Section 6.3.3.2, respectively) following the derivation of the key-wrapping key. This enables party V to provide assurance to party U that V has derived the correct key-wrapping key. A failure of key confirmation would alert party U that party V may not be able to unwrap the transported keying material, and the key-transport transaction could be discontinued before the keying material is wrapped and sent.

Key confirmation from party U (the intended key-transport sender) to party V (the intended key-transport receiver) can be incorporated in a C(2e, 2s), C(1e, 2s), or C(0e, 2s) scheme (as specified in Section 6.1.1.5.1, Section 6.2.1.5.1, or Section 6.3.3.1, respectively) prior to the key-transport portion of the transaction; in the case of a C(1e, 2s) or C(0e, 2s) scheme, party V would be required to contribute a nonce that is used as input to the key-derivation method when the key-warping key is derived. Key confirmation provided by party U enables party V to obtain assurance that party V has derived the same key that party U will employ to wrap the transported keying material. A failure of key confirmation may, for example, prompt party V to discontinue the current key-transport transaction (without attempting to unwrap any transported keying material) and notify party U that they must try again to establish a shared key-wrapping key.

As specified in Section 7.2, key confirmation can also be performed following the transport of the wrapped keying material, allowing party U to obtain assurance that party V has successfully employed the derived key-wrapping key to unwrap the transported keying material. Confirming party V's success in unwrapping the transported keying material also confirms that party V has correctly derived the key-wrapping key during the key-agreement portion of the transaction. Therefore, at the risk of transporting keying material that cannot be unwrapped, key confirmation following the transport of wrapped keying material (as specified in Section 7.2) provides an alternative to incorporating key confirmation (from party V to party U) in the key-agreement scheme.

The use of a C(1e, 2s), C(1e, 1s) or C(0e, 2s) scheme in DLC-based key transport allows for one-pass implementations of key transport (in cases where key confirmation is not required). Assuming that party V's static public key has been obtained previously, party U can include the wrapped keying material and all of the data required for party V to derive the key-wrapping key in a single message. On the other hand, the use of a C(2e, 2s) scheme necessitates the exchange

of two or more messages, since each party must (at least) provide an ephemeral public key to the other party in the transaction.

There are additional considerations that apply to the broadcast scenario, in which one sender transports the same keying material "simultaneously" (or within a short period of time) to multiple receivers for use following the key-transport transaction(s).

As noted in Section 7.1, this Recommendation's general prohibition against the reuse of an ephemeral key pair is relaxed in broadcast scenarios, permitting (but not requiring) the key-transport sender (acting as party U in the key-agreement portion of the transaction) to use the same ephemeral key pair when establishing key-wrapping keys with the multiple key-transport receivers. However, parties must proceed with caution when engaging in such practices (see, for example, "On Reusing Ephemeral Keys in Diffie-Hellman Key Agreement Protocols," by A. Menezes and B. Ustaoglu; available at http://cacr.uwaterloo.ca/techreports/2008/cacr2008-24.pdf).

As part of the proper implementation of this Recommendation, party U **should not** reuse an ephemeral public key when establishing key-wrapping keys for key transport in a broadcast scenario unless they and/or agents trusted to act on their behalf have determined the conditions (including the choice of key-agreement scheme) under which this practice meets the security requirements of both the sender (i.e., party U) and the receivers.

If, in a broadcast scenario, party U requires multiple receivers to provide evidence that they have successfully unwrapped the keying material sent to them, using key confirmation as specified in Section 7.2, it is imperative for party U to transport a unique MAC key to each receiver (as required by this Recommendation). In the absence of the compromise of any key-wrapping keys, this will deter one receiver from masquerading as another when returning a key-confirmation *MacTag* to party U (the key-transport sender).

9. Key Recovery

For some applications, the secret keying material used to protect data may need to be recovered (for example, if the normal reference copy of the secret keying material is lost or corrupted). In this case, either the secret keying material or sufficient information to reconstruct the secret keying material needs to be available (for example, 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.
2. An ephemeral public key **may** be saved.
3. An ephemeral private key **shall** be destroyed after use and, therefore, **shall not** be recoverable.
4. A symmetric key **may** be saved.

Note: This implies that keys derived from schemes where both parties generate ephemeral key pairs (i.e., the C(2e, 2s) and C(2e, 0s) schemes) cannot be made recoverable by reconstruction of the secret keying material by parties requiring the ephemeral private key in their calculations. For those schemes where only party U generates an ephemeral key pair (i.e., the C(1e, 2s) and C(1e, 1s schemes), only party V can recover the secret keying material by reconstruction.

General guidance on key recovery and the protections required for each type of key is provided in the Recommendation for Key Management [SP 800-57].

10. Implementation Validation

When the NIST Cryptographic Algorithm Validation Program (CAVP) and the Cryptographic Module Validation Program (CMVP) have established a validation program for this Recommendation, a vendor **shall** have its implementation tested and validated by the CAVP and CMVP in order to claim conformance to this Recommendation. Information on the CMVP is available at http://csrc.nist.gov/cryptval/.

An implementation claiming conformance to this Recommendation **shall** include one or more of the following capabilities:

- Domain parameter generation as specified in Section 5.5.1.
- Explicit domain parameter validation as specified in Section 5.5.2, item 2.
- Key pair generation as specified in Section 5.6.1; documentation **shall** include how assurance of domain parameter validity is expected to be achieved by the key pair owner.

- Explicit public key validation as specified in Section 5.6.2.3.1 for FFC or as specified in Sections 5.6.2.3.2 or 5.6.2.3.3 for ECC.
- A key-agreement scheme from Section 6, together with an **approved** key derivation method from Section 5.8 or an application-specific KDF from SP 800-135. Other key derivation methods may be temporarily allowed for backward compatibility; these other allowable methods and any restrictions on their use will be specified in FIPS 140-2 Annex D. If key confirmation is also claimed, the appropriate key confirmation technique from Section 5.9 **shall** be used. Documentation **shall** include how assurance of private key possession and assurance of domain parameter and public key validity are expected to be achieved by both the owner and the recipient.
- A key transport scheme as specified in Section 7.

An implementer **shall** also identify the appropriate specifics of the implementation, including:

- The security strength(s) of supported cryptographic algorithms; this will determine the parameter set requirements (see Table 1 and Table 2 in Section 5.5.1),
- The domain parameter generation method (see Section 5.5.1).
- The hash function (see Section 5.1),
- The MAC key length(s) (see Table 8 and Table 9 in Section 5.9.3),
- The MAC length(s) (see Table 8 and Table 9 in Section 5.9.3).
- The type of cryptography: FFC or ECC,
- The key-establishment schemes available (see Section 6),
- The key derivation method to be used, including the format of *OtherInfo* (see Section 5.8),
- The type of nonces to be generated (see Section 5.4),
- The NIST- Recommended elliptic curve(s) available (if appropriate),
- The key wrap algorithm, and
- The key confirmation scheme (see Section 5.9).

Appendix A: References (Informative)

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

[FIPS 140-2 IG] FIPS 140-2 Implementation Guidance, Available at http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf .

[FIPS 180] FIPS 180-4, Secure Hash Standard, February 2011.

[FIPS 186] FIPS 186-3, Digital Signature Standard, June 2009.

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

[FIPS 198] FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008.

[SP 800-38B] NIST SP 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005.

[SP 800-38F] DRAFT Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, August 11, 2011.

[SP 800-56C] Recommendation for Key Derivation through Extraction-then-Expansion, November 2010.

[SP 800-57] NIST SP 800-57, Recommendation for Key Management, August 2005.

[SP 800-90A] Recommendation for Random Number Generation Using Deterministic Random Bit Generators, January 2012.

[SP 800-108] Recommendation for Key Derivation Using Pseudorandom Functions, October 2009.

[SP 800-135] Recommendation for Existing Application-Specific Key Derivation Functions, December 2010.

[ANS X9.42] ANS X9.42-2003, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography

[ANS X9.62] ANS X9.62-2005, Key Cryptography for the Financial Services Industry: Elliptic Curve Digital Signature Algorithm (ECDSA).

[ANS X9.63] ANS X9.63-2011, Key Cryptography for the Financial Services Industry: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

[BM 1998] S. Blake-Wilson, A. Menezes, Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol, Technical Report CORR 98-42, University of Waterloo, 1998. Available at: http://cacr.math.uwaterloo.ca.

[CMU 2009] S. Chatterjee, A. Menezes, and B. Ustaoglu,Reusing Static Keys in Key Agreement Protocols, INDOCRYPT 2009, LNCS Vol. 5922, pp. 39–56,

Springer-Verlag, 2009. Available at: http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-36.pdf .

[CBH 2005] K. R. Choo, C. Boyd, and Y. Hitchcock, On Session Key Construction in Provably-Secure Key Establishment Protocols, LNCS, Vol. 3715, pp. 116-131, Springer-Verlag, 2005. Extended version available at: http://eprint.iacr.org/2005/206.pdf.

[Menezes 2007] A. Menezes, Another look at HMQV. Journal of Mathematical Cryptology, Vol.1(1), pp. 47-64, 2007

[RBB 2001] P. Rogaway, M. Bellare, D. Boneh, Evaluation of Security Level of Cryptography: ECMQVS (from SEC 1), Jan. 2001. Available at: http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1069_ks-ecmqv.pdf.

Appendix B: Rationale for Including Identifiers in the KDF Input

It is strongly recommended that identifiers for both parties to a key-agreement transaction be included among the data input to the key-derivation method – as a simple and efficient means of binding those identifiers to the derived keying material. (See Sections 5.8.)

The inclusion of sufficiently-specific identifiers for party U and party V provides assurance that the keying material derived by those parties will be different from the keying material that is derived by other parties (or by the same parties acting in opposite roles). As a result, key-agreement schemes gain resilience against unknown-key-share attacks and/or other exploitation techniques that depend on some type of confusion over the role played by each party (e.g., party U versus party V). See, for example, references [CBH 2005], [Menezes 2007], [RBB 2001], [BM 1998], and [CMU 2009], which all recommend the inclusion of identifiers in the key derivation method as a means of eliminating certain vulnerabilities.

In addition to identifiers, the inclusion of other context-specific information in the key-derivation input data can be used to draw finer distinctions between key-agreement transactions, providing assurance that parties will not derive the same keying material unless they agree on all of the included information. This can protect against attacks that rely on confusion concerning the context in which key-establishment takes place and/or how the derived keying material is to be used, see [CMU 2009]. Examples of additional context-specific information include (but are not limited to) the protocol employing the key-derivation method, protocol-defined session numbers, the key-agreement scheme that was employed to produce the shared secret Z, any ephemeral public keys and/or nonces exchanged during the key-agreement transaction, the bit length of the derived keying material, and its intended use.

Protocols employing an **approved** key-agreement scheme may employ alternative methods to bind participant identifiers (and/or other context-specific data) to the derived keying material or otherwise provide assurance that the participants in a key-agreement transaction share the same view of the context in which the keying material was established (including their respective roles and identifiers). However, this Recommendation makes no statement as to the adequacy of these other methods.

Appendix C: Data Conversions (Normative)

C.1 Integer-to-Byte String Conversion

Input: A non-negative integer C and the intended length n of the byte string satisfying

$$2^{8n} > C$$

When called from an FFC Scheme, $n = \lceil t / 8 \rceil$ bytes, where $t = \lceil \log_2 p \rceil$ where p is the large prime field order.

Output: A byte string S of length n bytes.

1. Let $S_1, S_2, \ldots, S_n$ be the bytes of S from leftmost to rightmost.
2. The bytes of S **shall** satisfy:

$$C = \sum 2^{8(n-i)} S_i \text{ for } i = 1 \text{ to } n.$$

C.2 Field-Element-to-Byte String Conversion

Input: An element α in the field F_q.

Output: A byte string S of length $n = \lceil t / 8 \rceil$ bytes, where $t = \lceil \log_2 q \rceil$.

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

C.3 Field-Element-to-Integer Conversion

Input: An element α in the field F_q.

Output: An integer x.

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

$$x = \sum 2^{(m-i)} s_i \qquad \text{for } i = 1 \text{ to } m.$$

Appendix D: Revisions (Informative)

The original version of this document was published in March, 2006. In March, 2007, the following revision was made to allow the dual use of keys during certificate requests:

In Section 5.6.4.2, the second item was originally as follows:

> "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 (for example, a digital signature key pair is not to be used for key establishment or vice versa)."

The item was changed to the following, where the changed text is indicated in italics:

> "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 (for example, a digital signature key pair is not to be used for key establishment or vice versa) *with the following possible exception: when requesting the (initial) certificate for a public static key-establishment key, the key establishment private key associated with the public key may be used to sign the certificate request. See SP 800-57, Part 1 on Key Usage for further information."*

In August 2012 version, the following revisions were made;

- Abstract – The March 2007 version cites ANS X9.42 and X9.63; this version directly provides information on the key establishment schemes (DH, MQV) and the underlying mathematics structure (discrete logs on finite field, elliptic curve).
- Section 3.1 – Added definitions of assumption, binding, bit string, byte, byte string, destroy, key-establishment pair, key wrapping key, trusted association; removed definitions on assurance of identifier, initiator, responder, (instead initiator and responder, all the schemes are defined in terms of party U and party V, see revision in Section 4), extended keying material to derived keying material (derived from the shared secret) and transported keying material (generated by the sendor in a key transport scheme.)
- Section 3.2 – The notations, C(ie), C(ie, js), MAC(*MacKey*, *MacData*), *MacTag*, *T*_bitlen(*X*), were introduced; the notation |x | is removed.
- Section 3.2 – Notations Z, Z_e, Z_s are used for both FFC and ECC and therefore moved up as general notations.
- Section 3.2 – The terms *GF(p)*, *GF(p)** were introduced for FFC.
- Section 4 – Used U and V to name the parties, rather than user the initiator and responder as the parties. Discussions about identifiers vs. identity and binding have been moved to Section 4.1.
- Section 4.1 – Added discussions on the concept of a trusted association;
- Section 5 – Table 1 in March 2007 version has been removed; the information is now provided in Tables 6 and 7 in Section 5.8.1, and Tables 8 and 9 in Section 5.9.3.

- Section 5.2 – Provided more details on MAC inputs (*MacKey* and *MacData*). Added text that MACs can be used for key derivation, as well as key confirmation. Added SP 800-38B (CMAC) as an **approved** MAC. Refers to the new Tables 6 and 7.
- Section 5.2.1 - *MacLen* now is a parameter, rather than an input variable. Refers to new Tables 8 and 9, instead of old Tables 1 and 2. Discusses the truncation of the MAC output.
- Section 5.4 – More discussion has been added about the use of nonces, including new requirements and recommendations.
- Section 5.5.1.1 – More information is provided about finite fields. Added the requirement that the leftmost bit of p and q be a 1. Table 1 has been shortened to address just the values of p and q; information about the hash function is now provided in Tables 6 and 7 of Section 5.8.1, and in Tables 8 and 9 of Section 5.9.3.
- Section 5.5.1.2 – More information is provided about elliptic curves. More details are provided on parameter values. Table 2 has been shorted to just address n and h; information about the hash function is now provided in Tables 6 and 7 of Section 5.8.1, and in Tables 8 and 9 of Section 5.9.3.
- Section 5.5.2 – A note about parameters generated by using SHA-1has been removed. The validation methods are referred to other documents (FIPS 186 and ANS X9.62). It is not a right place for such statement.
- Section 5.6 has been reorganized to make it clearer to understand key generation and obtaining the required assurances.
- Section 5.6.1.1 – FFC key-pair generation has been revised to require a randomly selected integer in the range in the range [2, q–2], rather than requiring a private key for FFC key pair generation to be unpredictable and generated by an **approved** RNG. Generation in accordance with FIPS 186-3 (as referenced therein) fulfills these requirements.
- Section 5.6.1.2 – ECC key-pair generation has been revised to require a randomly selected integer in the range in the range [2, n–2], rather than requiring a private key for ECC key pair generation to be unpredictable and generated by an **approved** RNG. Generation in accordance with FIPS 186-3 (as referenced therein) fulfills these requirements.
- New Section 5.6.2 – Discusses assurances and why they are required. Added Tables 3, 4, and 5 which summarize types of assurance.
- New Section 5.6.2.1 – Discusses the assurances required by a key-pair owner about its own key pair, including owner assurance of correct generation, static and ephemeral public-key validity, pair-wise consistency and private-key possession.
- New Section 5.6.2.2 – Discusses the assurances required by a public-key recipient, including static and ephemeral public-key validity, and static and ephemeral private-key possession.

- New Sections 5.6.3.2 and 5.6.3.3 – Different requirements are included for static and ephemeral key pairs. Included a case that an agent may act on behalf of a system user.
- Section 5.7 – Added requirements to destroy all values if there is an error and to destroy intermediate calculations have been added for each FFC and ECC primitive. Conversion calls have been added to convert to a string. Note that this removed such statements for the action steps for each scheme in Section 6.
- Section 5.8 – Key derivation has been divided into single-step key derivation methods (Section 5.8.1), an extract-then-expand key-derivation procedure (Section 5.8.2) and application-specific key-derivation methods (Section 5.8.3).
- Section 5.8.1 – Instead of using a hash function, the one-step method is now defined with a function H, which can be a hash function or an HMAC with an approved hash function. Added tables defining minimum length for the hash functions with regard to each parameter set;; and added more complete discussions about *OtherInfo*, including the concatenation and ASN.1 formats included in the previous version. HMAC with an **approved** hash function is now **approved** for key derivation, in addition to the hash function specified in the previous version.
- Section 5.8.2 – Added reference to an **approved** two-step method – an extraction-then-expansion method – that is specified in SP 800-56C.
- Section 5.8.3 – Added reference to the application-specific key derivation methods provided in SP 800-135.
- Moved general introduction of key confirmation to Section 5.9 – Incorporates the material from Section 8 (with additional introductory material).
- New Section 5.9.1.1 – Emphasizes more clearly that a nonce is required if there is no ephemeral key; added guidance on what to do if key confirmation fails.
- New Section 5.9.2 – Emphasizes that if no ephemeral key is used, then a nonce is required.
- New Section 5.9.3 – Discussions about the security strength of the MacTag are provided, along with tables on the minimum *MacKey* length and *MacLen* values.
- Section 6 – The notation C(ie) replaces C(i), and C(ie, js) replaces C(i, j). If U does not contribute a static key, then the requirement for a non-null identifier is now transaction dependent, rather than required. Rationale for choosing the C(ie, js) schemes has been moved to a new Section 8, instead of after each class of schemes. Assumptions are specified for each type of scheme, rather than prerequisites.
- Section 6.1.1 (and similarly for Sections 6.2.1, 6.2.2 and 6.3) –Added a new assumption that if an identifier is used as a label, then the identifier must have a trusted association to that party's static key. The discussion on the need for a trusted association has been added.
- Section 6.1.1.1 (dhHybrid1) – More guidance is provided about error handling. Specifically allows the reuse of an ephemeral key pair in a broadcast scenario. This is also provided in Sections 6.1.1.2, 6.1.1.3 and 6.1.1.4.

- New Section 6.1.1.5 (and similarly in new Sections 6.1.2.3, 6.2.1.5, 6.2.2.3 and 6.3.3) – Key confirmation is incorporated to each applied subcategory of schemes. This material was previously provided in Section 8.4 of the previous version.
- Section 6.2.1 (C(1e,2s) schemes) – Added additional assumptions which were included in the previous prerequisites. This includes obtaining assurance of static public key validity and private keys possession of the key-pair owner.
- Section 7 – Has been revised to specify DLC-based key-agreement and key transport in the same key-establishment transaction, with party U acting as the key-transport sender. In addition, optional key confirmation from party V to party U following the key-transport process has been specified.
- Section 8 – The rationale for choosing each scheme type has been moved from Section 6 of the previous version. A new section on the rationale associated with key-transport has been included.
- All figures are replaced to reflect the content, text, and terminology changes.
- Old Appendix A, Summary of Differences between this Recommendation and ANS X9 Standards, was removed. Note that X9.42 was withdrawn, while X9.63 has modified to be consistent with this Recommendation.
- Appendix B – The requirement of including identifiers as part of the *OtherInfo* is replaced with text that. it is strongly recommended that identifiers for both parties to a key-agreement transaction be included among the data input to a key-derivation method. A paragraph has been added stating that there may be other ways to bind identifiers to derived keying material, but the recommendation makes no statement on the adequacy of this.
- The new Appendix A includes all the informative references, which was in Appendix D in March 2007 version.
- The old Appendix E becomes Appendix D and the changes on March 2007 version are added as listed here.