Publication Number: **NIST Special Publication (SP) 800-56A Revision 3**

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

Publication Date: **April 2018**

- Final Publication: https://doi.org/10.6028/NIST.SP.800-56Ar3 (which links to https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf).
- Related Information on CSRC:
  Final: https://csrc.nist.gov/publications/detail/sp/800-56A/rev-3/final

National Institute of Standards and Technology • U.S. Department of Commerce

Draft NIST Special Publication 800-56A
Revision 3

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

Elaine Barker
Lily Chen
Sharon Keller
Allen Roginsky
Apostol Vassilev
Richard Davis

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

National Institute of
Standards and Technology
U.S. Department of Commerce

Draft NIST Special Publication 800-56A
Revision 3

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

Elaine Barker
Lily Chen
Sharon Keller
Allen Roginsky
Apostol Vassilev
Computer Security Division
Information Technology Laboratory

Richard Davis
National Security Agency

August 2017

U.S. Department of Commerce
Wilbur L. Ross, Jr., Secretary

National Institute of Standards and Technology
Kent Rochford, Acting NIST Director and Under Secretary of Commerce for Standards and Technology

Authority

This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C. § 3551 *et seq.*, Public Law (P.L.) 113-283. NIST is responsible for developing information security standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130.

Nothing in this publication should be taken to contradict the 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. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

National Institute of Standards and Technology Special Publication 800-56A Revision 3
Natl. Inst. Stand. Technol. Spec. Publ. 800-56A Rev. 3, 170 pages (August 2017)
CODEN: NSPUE2

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by Federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, Federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. Many NIST cybersecurity publications, other than the ones noted above, are available at http://csrc.nist.gov/publications.

Public comment period: *August 7, 2017* through *November 6, 2017*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: SP800-56a_comments@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in Federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

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 Menezes-Qu-Vanstone (MQV) key establishment schemes.

Keywords

Diffie-Hellman; elliptic curve cryptography; finite field cryptography; key agreement; key confirmation; key derivation; key establishment; key transport; MQV.

Acknowledgements

The authors gratefully acknowledge the contributions on previous versions of this document by Mike Hopper, Don Johnson, Laurie Law, and Miles Smid.

Conformance Testing

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

Notes to Reviewers

Significant changes in this revision of SP 800-56A include:

1. The approval of specific safe-prime groups and the associated "safe" FFC domain parameters (see Section 5.5.1.1). These groups are named in Appendix E. The previously defined FFC parameter-size sets, FB and FC, are now referred to as "FIPS 186-type" parameter-size sets. (Parameter-size set FA is no longer approved for use.)
2. ECC parameter-size sets are no longer identified (see Section 5.5.1.2), **Approved** ECC domain parameters will be those associated with either the recommended elliptic curves now found in FIPS 186-4 or (eventually) other specifically **approved** elliptic curves, which will be named in a future publication: SP 800-186. The specifications of the elliptic curves now found in FIPS 186-4 will be moved to SP 800-186.
3. Routines for generating FFC and ECC key pairs have been added to the document instead of referring to the key-pair generation routines in FIPS 186-4 (see Section 5.6.1). The included FFC routines permit some flexibility in the generation of FFC key pairs associated with safe-prime groups, but retain the FIPS 186-specified methods for generating FFC key pairs using FIPS 186-type domain parameters. The FIPS 186-specified methods for generating ECC key pairs are also included.
4. When using an **approved** safe-prime group for key-establishment purposes, assurance of another party's possession of the private key corresponding to a received static public key **shall** be obtained by the recipient either directly, by engaging in a key-agreement transaction as specified in Section 5.6.2.2.3.2, or indirectly, from a trusted third party (e.g., a CA) who has obtained the assurance directly. Assurance of possession of the FIPS 186-type domain parameters (specified in Section 5.5.1.1 and in the previous version of this Recommendation) may also by initially obtained using the private key to sign a certificate request (see Section 5.6.3.2). However, the provision of a signed certificate request to a CA (or any other signature-based technique) is **not approved** as a means of providing assurance of private-key possession when the static public key is an element of an **approved** safe-prime group.
5. A simple partial public-key validation will be permitted for ephemeral FFC public keys selected from an approved safe-prime group (see Section 5.6.2.3.2).
6. A more detailed list of revisions is provided at the end of Appendix D.

Questions:

1. Is there a case to be made for using elliptic curves defined over $GF(2^m)$? If not, is there any objection to restricting ECC key-agreement schemes to the use of elliptic curves defined over $GF(p)$, where p is an odd prime?
2. Which of the currently approved key-agreement schemes are actually used (and by what protocols)? Are there any schemes in Section 6 that should no longer be

approved for use (e.g., FFC MQV, which is specified in Sections 6.1.1.3 and 6.2.1.3)?

3. Should Section 7 be removed, expanded or reduced in content? Two versions of Section 7 are provided for your consideration. Please compare with the current version (revision 2) and tell us what would be preferred. Revision 2 is available at:

   http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

4. Are the FIPS 186-type domain parameters **actually being used** anywhere (rather than just available in an implementation in order to be validated)?

Table of Contents

1. Introduction **1**

2. Scope and Purpose **1**

3. Definitions, Symbols and Abbreviations **2**

3.1 Definitions....2

3.2 Symbols and Abbreviations....7

4. Overview of Key-Establishment Schemes **13**

4.1 Key Establishment Preparations....14

4.2 Key-Agreement Process....16

4.3 DLC-based Key-Transport Process....18

5. Cryptographic Elements **20**

5.1 Cryptographic Hash Functions....20

5.2 Message Authentication Code (MAC) Algorithm....20

5.2.1 MAC Tag Computation for Key Confirmation....20

5.2.2 MAC Tag Verification for Key Confirmation....21

5.3 Random Number Generation....21

5.4 Nonce....21

5.5 Domain Parameters....22

5.5.1 Domain-Parameter Selection/Generation....23

5.5.1.1 FFC Domain Parameter Selection/Generation....23

5.5.1.2 ECC Domain-Parameter Selection....24

5.5.2 Assurances of Domain-Parameter Validity....25

5.5.3 Domain Parameter Management....26

5.6 Key-Establishment Key Pairs....26

5.6.1 Key-Pair Generation....26

5.6.1.1 FFC Key-Pair Generation....27

5.6.1.1.1 Using the Approved Safe-Prime Groups....27

5.6.1.1.2 Using the FIPS 186-Type FFC Parameter-size Sets....27

5.6.1.1.3 Key-Pair Generation Using Extra Random Bits....27

5.6.1.1.4 Key-Pair Generation by Testing Candidates................28
5.6.1.2 ECC Key-Pair Generation...29
5.6.1.2.1 Key Pair Generation Using Extra Random Bits 30
5.6.1.2.2 Key Pair Generation by Testing Candidates................31
5.6.2 Required Assurances..32
5.6.2.1 Assurances Required by the Key Pair Owner.............................33
5.6.2.1.1 Owner Assurance of Correct Generation.....................34
5.6.2.1.2 Owner Assurance of Private-Key Validity 35
5.6.2.1.3 Owner Assurance of Public-Key Validity 35
5.6.2.1.4 Owner Assurance of Pair-wise Consistency................36
5.6.2.1.5 Owner Assurance of Possession of the Private Key 37
5.6.2.2 Assurances Required by a Public Key Recipient........................37
5.6.2.2.1 Recipient Assurance of Static Public-Key Validity.....39
5.6.2.2.2 Recipient Assurance of Ephemeral Public-Key Validity 39
5.6.2.2.3 Recipient Assurance of the Owner's Possession of a Static Private Key 39
5.6.2.2.4 Recipient Assurance of the Owner's Possession of an Ephemeral Private Key 41
5.6.2.3 Public Key Validation Routines...42
5.6.2.3.1 FFC Full Public-Key Validation Routine 43
5.6.2.3.2 FFC Partial Public-Key Validation Routine 43
5.6.2.3.3 ECC Full Public-Key Validation Routine....................44
5.6.2.3.4 ECC Partial Public-Key Validation Routine................45
5.6.3 Key Pair Management..46
5.6.3.1 Common Requirements on Static and Ephemeral Key Pairs......46
5.6.3.2 Specific Requirements on Static Key Pairs 46
5.6.3.3 Specific Requirements on Ephemeral Key Pairs 47
5.7 DLC Primitives 47
5.7.1 Diffie-Hellman Primitives 48

5.7.1.1 Finite Field Cryptography Diffie-Hellman (FFC DH) Primitive 48
5.7.1.2 Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive 49
5.7.2 MQV Primitives 49
5.7.2.1 Finite Field Cryptography MQV (FFC MQV) Primitive 49
5.7.2.1.1 MQV2 Form of the FFC MQV Primitive 50
5.7.2.1.2 MQV1 Form of the FFC MQV Primitive 51
5.7.2.2 ECC MQV Associate Value Function 51
5.7.2.3 Elliptic Curve Cryptography MQV (ECC MQV) Primitive 51
5.7.2.3.1 Full MQV Form of the ECC MQV Primitive 52
5.7.2.3.2 One-Pass Form of the ECC MQV Primitive 52
5.8 Key-Derivation Methods for Key-Agreement Schemes 53
5.8.1 Performing the Key Derivation 53
5.8.2 FixedInfo 54
5.8.2.1 One-step Key Derivation 55
5.8.2.1.1 The Concatenation Format for *FixedInfo* 56
5.8.2.1.2 The ASN.1 Format for *FixedInfo* 57
5.8.2.2 Two-step Key-Derivation (Extraction-then-Expansion) 57
5.8.2.3 Other Formats for *FixedInfo* 57
5.9 Key Confirmation 57
5.9.1 Unilateral Key Confirmation for Key-Agreement Schemes 58
5.9.2 Bilateral Key Confirmation for Key-Agreement Schemes 61
5.9.3 Selecting the MAC and Other Key-Confirmation Parameters 61
6. Key Agreement 64
6.1 Schemes Using Two Ephemeral Key Pairs, C(2e) 66
6.1.1 C(2e, 2s) Schemes 67
6.1.1.1 dhHybrid1, C(2e, 2s, FFC DH) Scheme 68
6.1.1.2 (Cofactor) Full Unified Model, C(2e, 2s, ECC CDH) Scheme ..70
6.1.1.3 MQV2, C(2e, 2s, FFC MQV) Scheme 72

6.1.1.4 Full MQV, C(2e, 2s, ECC MQV) Scheme 74
6.1.1.5 Incorporating Key Confirmation into a C(2e, 2s) Scheme 75
6.1.1.5.1 C(2e, 2s) Scheme with Unilateral Key Confirmation Provided by Party U to Party V 75
6.1.1.5.2 C(2e, 2s) Scheme with Unilateral Key Confirmation Provided by Party V to Party U 76
6.1.1.5.3 C(2e, 2s) Scheme with Bilateral Key Confirmation 77
6.1.2 C(2e, 0s) Schemes....78
6.1.2.1 dhEphem, C(2e, 0s, FFC DH) Scheme 79
6.1.2.2 (Cofactor) Ephemeral Unified Model, C(2e, 0s, ECC CDH) Scheme 81
6.1.2.3 Key Confirmation for C(2e, 0s) Schemes....82
6.2 Schemes Using One Ephemeral Key Pair, C(1e) Schemes 82
6.2.1 C(1e, 2s) Schemes....82
6.2.1.1 dhHybridOneFlow, C(1e, 2s, FFC DH) Scheme 84
6.2.1.2 (Cofactor) One-Pass Unified Model, C(1e, 2s, ECC CDH) Scheme 86
6.2.1.3 MQV1, C(1e, 2s, FFC MQV) Scheme 89
6.2.1.4 One-Pass MQV, C(1e, 2s, ECC MQV) Scheme....91
6.2.1.5 Incorporating Key Confirmation into a C(1e, 2s) Scheme 93
6.2.1.5.1 C(1e, 2s) Scheme with Unilateral Key Confirmation Provided by Party U to Party V 93
6.2.1.5.2 C(1e, 2s) Scheme with Unilateral Key Confirmation Provided by Party V to Party U 94
6.2.1.5.3 C(1e, 2s) Scheme with Bilateral Key Confirmation 95
6.2.2 C(1e, 1s) Schemes....96
6.2.2.1 dhOneFlow, C(1e, 1s, FFC DH) Scheme 98
6.2.2.2 (Cofactor) One-Pass Diffie-Hellman, C(1e, 1s, ECC CDH) Scheme 100
6.2.2.3 Incorporating Key Confirmation into a C(1e, 1s) Scheme 102
6.2.2.3.1 C(1e, 1s) Scheme with Unilateral Key Confirmation Provided by Party V to Party U 102

6.3 C(0e, 2s) Schemes 103
6.3.1 dhStatic, C(0e, 2s, FFC DH) Scheme 105
6.3.2 (Cofactor) Static Unified Model, C(0e, 2s, ECC CDH) Scheme 107
6.3.3 Incorporating Key Confirmation into a C(0e, 2s) Scheme 108
6.3.3.1 C(0e, 2s) Scheme with Unilateral Key Confirmation Provided by Party U to Party V 109
6.3.3.2 C(0e, 2s) Scheme with Unilateral Key Confirmation Provided by Party V to Party U 110
6.3.3.3 C(0e, 2s) Scheme with Bilateral Key Confirmation 111
7. DLC-Based Key Transport (Alternative 1) 113
7.1 Key Transport Scheme 114
7.1.1 Key-Wrapping using AES-CCM 115
7.1.2 Key-Unwrapping using AES-CCM 117
7.1.3 Key Wrapping Using KW or KWP 118
7.1.4 Key Unwrapping Using KW or KWP 119
7.2 Key Confirmation for Transported Keying Material 120
7. DLC-Based Key Transport (Alternative 2) 122
7.1 Assumptions 122
7.2 Key-Transport Scheme 123
7.3 Key Confirmation for Transported Keying Material 124
8. Rationale for Selecting a Specific Scheme 125
8.1 Rationale for Choosing a C(2e, 2s) Scheme 126
8.2 Rationale for Choosing a C(2e, 0s) Scheme 127
8.3 Rationale for Choosing a C(1e, 2s) Scheme 128
8.4 Rationale for Choosing a C(1e, 1s) Scheme 130
8.5 Rationale for Choosing a C(0e, 2s) Scheme 131
8.6 Choosing a Key-Agreement Scheme for use in Key Transport 132
9. Key Recovery 137
10. Implementation Validation 138
Appendix A: References 140

A.1 Normative References....................140
A.2 Informative References....................142
Appendix B: Rationale for Including Identifiers and other Context-specific Information in the KDM Input (Informative) 143
C.1 Integer-to-Byte String Conversion....................144
C.2 Field-Element-to-Byte String Conversion....................144
C.3 Field-Element-to-Integer Conversion....................145
C.4 Conversion of a Bit String to an Integer....................145
Appendix D: Revisions (Informative) 146
Appendix E: Approved ECC Curves and FCC Safe-prime Groups 154

List of Figures

Figure 1: Owner key-establishment preparations 16

Figure 2: Key-agreement process. 17

Figure 3: Key-transport process....19

Figure 4: C(2e, 2s) schemes: each party contributes a static and an ephemeral key pair.....67

Figure 5: C(2e, 2s) scheme with unilateral key confirmation from party U to party V........76

Figure 6: C(2e, 2s) scheme with unilateral key confirmation from party V to party U.......77

Figure 7: C(2e, 2s) scheme with bilateral key confirmation....78

Figure 8: C(2e, 0s) schemes: each party contributes only an ephemeral key pair................79

Figure 9: C(1e, 2s) schemes: party U contributes a static and an ephemeral key pair while party V contributes only a static key pair....83

Figure 10: C(1e, 2s) scheme with unilateral key confirmation from party U to party V......94

Figure 11: C(1e, 2s) scheme with unilateral key confirmation from party V to party U......95

Figure 12: C(1e, 2s) scheme with bilateral key confirmation....96

Figure 13: C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair....97

Figure 14: C(1e, 1s) scheme with unilateral key confirmation from party V to party U....103

Figure 15: C(0e, 2s) schemes: each party contributes only a static key pair......................104

Figure 16: C(0e, 2s) scheme with unilateral key confirmation from party U to party V....109

Figure 17: C(0e, 2s) scheme with unilateral key confirmation from party V to party U....110

Figure 18: C(0e, 2s) scheme with bilateral key confirmation....111

List of Tables

Table 1: FIPS 186-type FFC parameter-size sets....24
Table 2: Initial assurances required by the key-pair owner....33
Table 3: Optional renewal of assurances by the key-pair owner....34
Table 4: Optional renewal of assurances by the key-pair owner....38
Table 5: Approved MAC algorithms....61
Table 6: Key-agreement scheme categories....63
Table 7: Key-agreement scheme subcategories....63
Table 8: Key-agreement schemes....64
Table 9: dhHybrid1 key-agreement scheme summary....68
Table 10: Full unified model key-agreement scheme summary....70
Table 11: MQV2 key-agreement scheme summary....72
Table 12: Full MQV key-agreement scheme summary....74
Table 13: dhEphem key-agreement scheme summary....79
Table 14: Ephemeral unified model key-agreement scheme....80
Table 15: dhHybridOneFlow key-agreement scheme summary....85
Table 16: One-pass unified model key-agreement scheme summary....87
Table 17: MQV1 Key-agreement scheme summary....89
Table 18: One-pass MQV model key-agreement scheme summary....92
Table 19: dhOneFlow key-agreement scheme summary....99
Table 20: One-pass Diffie-Hellman key-agreement scheme summary....101
Table 21: dhStatic key-agreement scheme summary....105
Table 22: Static unified model key-agreement scheme summary....107

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, 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 (SP) 800-57, *Recommendation for Key Management* [SP 800-57]. This Recommendation (i.e., SP 800-56A) and SP 800-57 are intended to provide sufficient information for a vendor to implement secure key establishment using asymmetric algorithms in 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

AES-CCM	The CCM block cipher mode specified in SP 800-38C for the AES algorithm specified in FIPS 197 for key sizes of either 128, 192 or 256 bits.
AES-CMAC	The CMAC block cipher mode specified in SP 800-38B for the AES algorithm specified in FIPS 197 for key sizes of either 128, 192 or 256 bits.
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	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.
Big-endian	The property of a byte string having its bytes positioned in order of decreasing significance. In particular, the leftmost (first) byte is the most significant byte (containing the most significant eight bits of the corresponding bit string) and the rightmost (last) byte is the least significant byte (containing the least significant eight bits of the corresponding bit string). For the purposes of this Recommendation, it is assumed that the bits within each byte of a big-endian byte string are also positioned in order of decreasing significance (beginning with the most significant bit in the leftmost position and ending with the least significant bit in the rightmost position).
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. Also known as a binary string.

Byte	A bit string consisting of eight bits.
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.
Critical security parameter (CSP)	Security-related information whose disclosure or modification can compromise the security of a cryptographic module. Domain parameters, secret or private keys, shared secrets, key-derivation keys, intermediate values and secret salts are examples of quantities that may be considered CSPs in this Recommendation. See FIPS 140.
Cryptographic module	The set of hardware, software and/or firmware that implements **approved** security functions (including cryptographic algorithms and key generation). See FIPS 140.
Destroy	In this Recommendation, 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. Also known as *zeroization* in FIPS 140.
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.
Fresh	Newly established keying material that is statistically independent of any previously established keying material.
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.
Identifier	A bit string that is associated with a person, device or organization. It may be an identifying name or a nickname, or may be something more abstract (for example, a string consisting of an IP address).
Integrity	A property whereby data has not been altered in an unauthorized manner since it was created, transmitted or stored.
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) possesses the correct secret keying material and/or shared secret from which that keying material is derived.
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 function	A function used to derive keying material from a shared secret (or a key) and other information.
Key-derivation method	A method to derive keying material from a shared secret and other information. A key-derivation method may use a key-derivation function or a key-derivation procedure.
Key-derivation procedure	A multi-step process that uses an **approved** MAC algorithm to derive keying material 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 instance of establishing secret keying material using a key-agreement or key-transport transaction.

Key-transport	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	A method of protecting keying material (along with associated integrity information) that provides both confidentiality and integrity protection by using symmetric-key algorithms.
Key-wrapping key	In this Recommendation, a key-wrapping key is a symmetric key established during 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, for example, 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 (possibly by truncation) that can be used by an entity to verify the integrity and the origination of the information used as input to the MAC algorithm.
Message Authentication Code (MAC) algorithm	A family of cryptographic functions that is parameterized by a symmetric key. Each of the functions can act on input data (called a "message") of variable length to produce an output value of a specified length. The output value is called the MAC of the input message. An **approved** MAC algorithm is expected to satisfy the following property (for each of its supported security levels): It must be computationally infeasible to determine the (as yet unseen) MAC of a message without knowledge of the key, even if one has already seen the results of using that key to compute the MACs of other (different) messages. A MAC algorithm can be used to provide data-origin authentication and data-integrity protection. In this Recommendation, a MAC algorithm is used for key confirmation; the use of MAC algorithms for key derivation is addressed in SP 800-56C.
Nonce	A time-varying value that has at most an acceptably small 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 is authorized to use the ephemeral private key of the 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).
Random nonce	A nonce containing a random-value component that is generated anew for each nonce.
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 set of unambiguously specified transformations that provide 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.
Sender	The party that sends secret keying material to the receiver in a key-transport transaction. Contrast with receiver.
Shall	This term is used to indicate 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 during a key-establishment scheme, is known by both participants, and is used as input to a key-derivation method to produce keying material.
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.
Store-and-forward	**A** telecommunications technique in which information is sent to an intermediate station where it is kept and later sent to the final destination or to another intermediate station.
Symmetric-key algorithm	A cryptographic algorithm that uses a single secret key that is shared between authorized parties.
Targeted security strength	The maximum security strength that is intended to be supported by one or more implementation-related choices (such as algorithms, primitives, auxiliary functions, parameter sizes and/or actual parameters) for the purpose of instantiating a cryptographic mechanism. In this Recommendation, it is assumed that the targeted security strength of any instantiation of an **approved** key-establishment scheme has a value greater than or equal to 112 bits and less than or equal to 256 bits.
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 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(*ie*)	Notation for a category of key-establishment schemes in which i ephemeral key pairs are used, where $i \in \{0, 1, 2\}$.
C(*ie*, *js*)	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.
CSP	Critical Security Parameter.
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.
ID	The bit string denoting the identifier associated with an entity.
KC	Key Confirmation.
KDM	Key-Derivation Method.
KM	Keying Material.
KWK	Key-Wrapping Key.
len(x)	The bit length of the shortest base-two representation of the positive integer x, i.e., $\text{len}(x) = \lfloor \log_2(x) \rfloor + 1$.
MAC	Message Authentication Code.

MAC(*MacKey*, *MacData*)	A MAC algorithm with *MacKey* as the key, and *MacData* as the data.
MacTag	A MAC tag.
MacTagLen	The length of the *MacTag* in bits.
MQV	The Menezes-Qu-Vanstone key-agreement primitive.
Null	The empty bit string
RBG	Random Bit Generator.
SHA	Secure Hash Algorithm (as specified in FIPS 180 and FIPS 202).
$T_{bitLen}(X)$	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, $T_2(1011) = 10$, $T_3(1011) = 101$, $T_4(1011) = 1011$, and $T_5(1011) = 1011$.
TTP	Trusted Third Party.
U, V	Represents the two parties in a (pair-wise) key-establishment scheme.
{ }	In this Recommendation, the curly braces { } are used in the following three situations: (1) {*x*} is used to indicate that the inclusion of *x* is optional; for example, the notation "Input: *w* {, *x*}, *y*, and *z*" implies that the inclusion of *x* as an input is optional. (2) If both *X* and *Y* are binary strings, the notation of binary string "*Y*{‖*X*}" implies that the concatenation of string *X* is optional. (3) $\{x_1, x_2, \ldots, x_k\}$ indicates a set with elements $x_1, x_2, \ldots, x_k$.
$X \| Y$	The concatenation of two bit strings *X* and *Y*. For example, 11001 ‖ 010 = 11001010.
$[a, b]$	The set of integers *x*, such that $a \le x \le b$.
$\lceil x \rceil$	The ceiling of *x*; the smallest integer $\ge x$. For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$.
$\lfloor x \rfloor$	The floor of *x*; the greatest integer that does not exceed *x*. For example, $\lfloor 2.1 \rfloor = 2$, and $\lfloor 4 \rfloor = 4$.
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 $\{0, 1, \ldots, p-1\}$. 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. In short, $j = k - \lfloor k/p \rfloor p$.
p	An FFC domain parameter; an odd prime number that determines the size of the finite field $GF(p)$.
counter	An optional 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	When used as an FFC domain parameter, q is the (odd) prime number equal to the order of the multiplicative subgroup of $GF(p)^*$ generated by 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 interval $[1, q - 1]$. (In some instances, r_U, and/or r_V may be restricted to a subinterval of the form $[1, 2^N - 1]$; see Section 5.6.1.1.1.)
t_U, t_V	The ephemeral public keys of party U and party V, respectively. These are integers in the interval $[2, p - 2]$.

SEED	An FFC domain parameter; an initialization value that is used during domain parameter generation that can also be used later to provide assurance that the resulting domain parameters were generated using an **approved** process.
x_U, x_V	The static private keys of party U and party V, respectively. These are integers in the interval $[1, q-1]$. (In some instances, x_U, and/or x_V may be restricted to a subinterval of the form $[1, 2^N-1]$; see Section 5.6.1.1.1.)
y_U, y_V	The static public keys of party U and party V, respectively. These are integers in the interval $[2, p-2]$.

ECC:

a, *b*	ECC domain parameters; two elements in the finite field *GF*(*q*) that define the (Weierstrass) equation of an elliptic curve, $y^2 = x^3 + ax + b$ when *q* is an odd prime or $y^2 + xy = x^3 + ax^2 + b$ when $q = 2^m$ for some prime 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 interval $[1, n-1]$.
$d_{s,U}$, $d_{s,V}$	The static private keys of party U and party V, respectively. These are integers in the interval $[1, n-1]$.
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*.

$\emptyset$	The (additive) identity element of an elliptic curve group; also called the "neutral point" of that group. $\emptyset$ is the unique element satisfying $Q + \emptyset = \emptyset + Q = Q$ for each Q in the group. For the (Weierstrass) elliptic curve groups considered in this Recommendation, a special "point at infinity" serves as $\emptyset$.
q	When used as an ECC domain parameter, q is the field size. It is either an odd prime p, or 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 that is 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 that is 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 later to provide assurance that the resulting domain parameters were generated using an **approved** 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 given **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 party U and party 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, the key 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 several 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 the 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.

The security of a key-establishment scheme depends on its implementation, and this document includes several practical recommendations for implementers. For example, good security practice dictates that implementations of procedures employed by primitives, operations, schemes, etc. include steps that destroy any potentially sensitive locally stored data that is created (and/or copied for use) during the execution of a given procedure, and whose continued local storage is not required after the procedure has been exited. The destruction of such locally stored data ideally occurs prior to or during any exit from the procedure. This is intended to limit opportunities for unauthorized access to sensitive information that might compromise a key-establishment process and to prevent its use for any other purpose.

Explicit instructions for the destruction of certain potentially sensitive values that are likely to be locally stored by procedures are included in the specifications found in this Recommendation. Examples of such values include local copies of any portions of secret or private keys that are employed or generated during the execution of a procedure, intermediate results produced during computations, and locally stored duplicates of values that are ultimately output by a procedure. However, it is not possible to anticipate the form of all possible implementations of the specified primitives, operations, schemes, etc., making it impossible to enumerate all potentially sensitive data that might be locally stored by a procedure employed in a given implementation. Nevertheless, the destruction of any potentially sensitive locally stored data is an obligation of all implementations.

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, as specified in Section 5.5.1 from an **approved** list (see Appendix E) or (in the FFC case) generated as specified in Section 5.5 by a trusted party. These parameters will determine the type of arithmetic used to generate key pairs and compute shared secrets. The owner must have assurance of the validity of these 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 (providing source authentication and integrity protection for the entire key pair, and confidentiality protection of (at least) the private key). 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 "NIST123", rather than an identifier that names a given 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 to obtain its key pair(s) in preparation for key establishment.

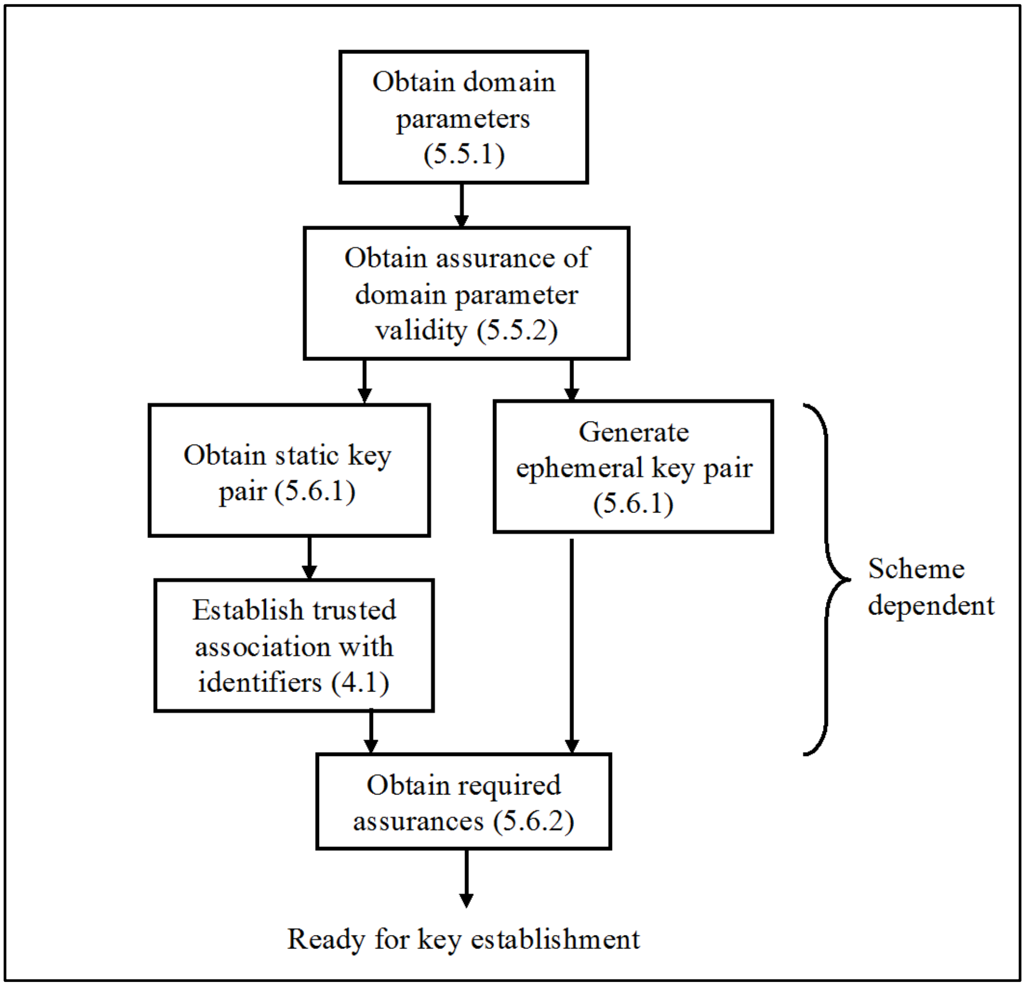

Figure 1: Owner key-establishment preparations

4.2 Key-Agreement Process

A key-agreement process specified in this Recommendation consists of a sequence of ordered steps. Figure 2 depicts the steps that may be required of an entity when establishing secret keying material with another entity using one of the key-agreement schemes described in this Recommendation. Some discrepancies in the order of the steps may occur, 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).

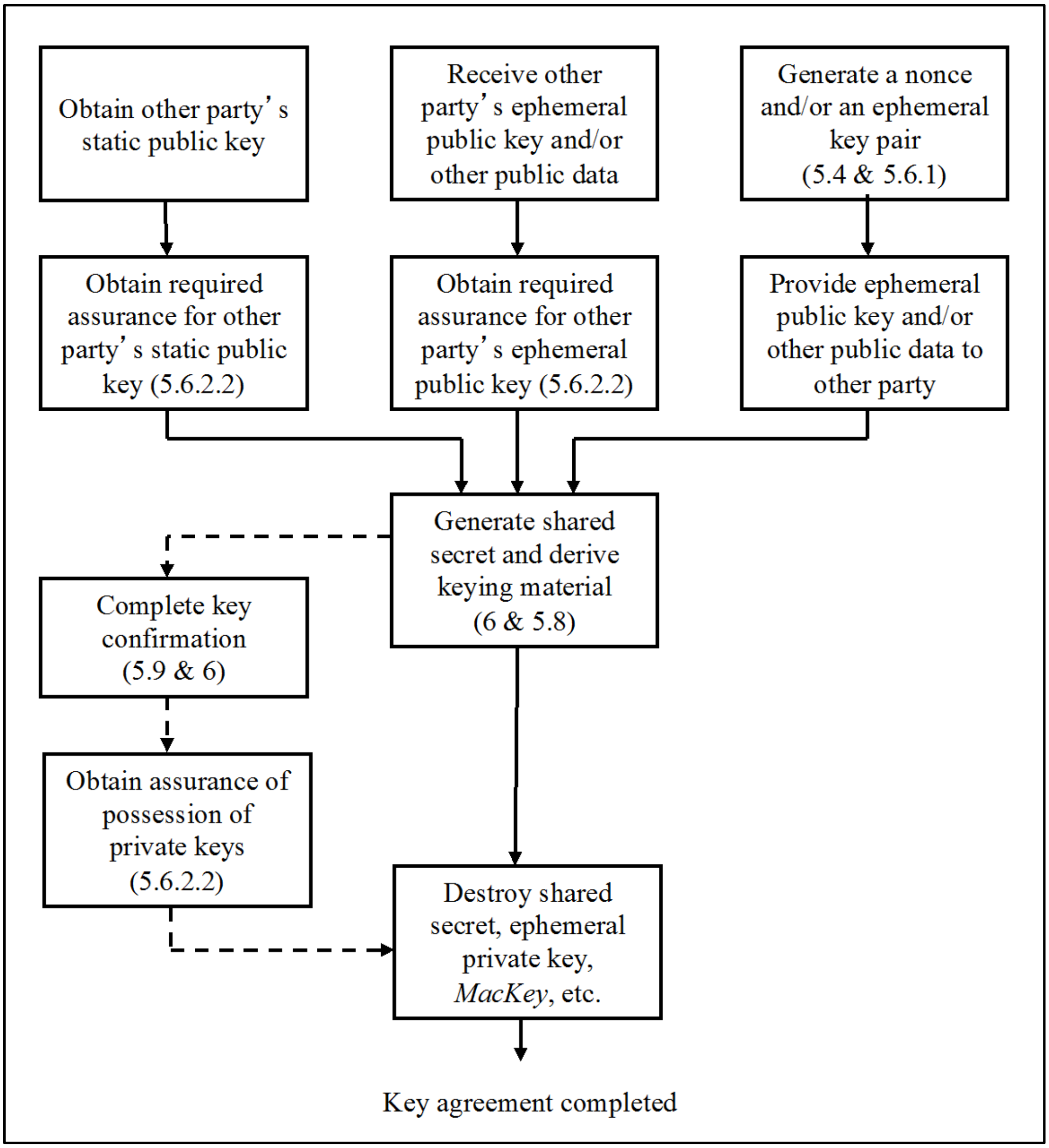

Figure 2: Key-agreement process.

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.

Note that some of the actions shown in Figure 2 may be absent from certain schemes. The specifications of this Recommendation indicate when an 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. (Note: this restriction above does not prohibit the use of derived keying material for key confirmation performed *during* the key-agreement transaction.) See Section 5.6.2.2.3 for **approved** methods for obtaining this assurance.

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 party.

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 party.

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 party.

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.

4.3 DLC-based Key-Transport Process

The key-transport 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 party to another using a key-transport scheme; the preceding key-agreement portion of the transaction is discussed in Section 4.2 and shown in Figure 2.

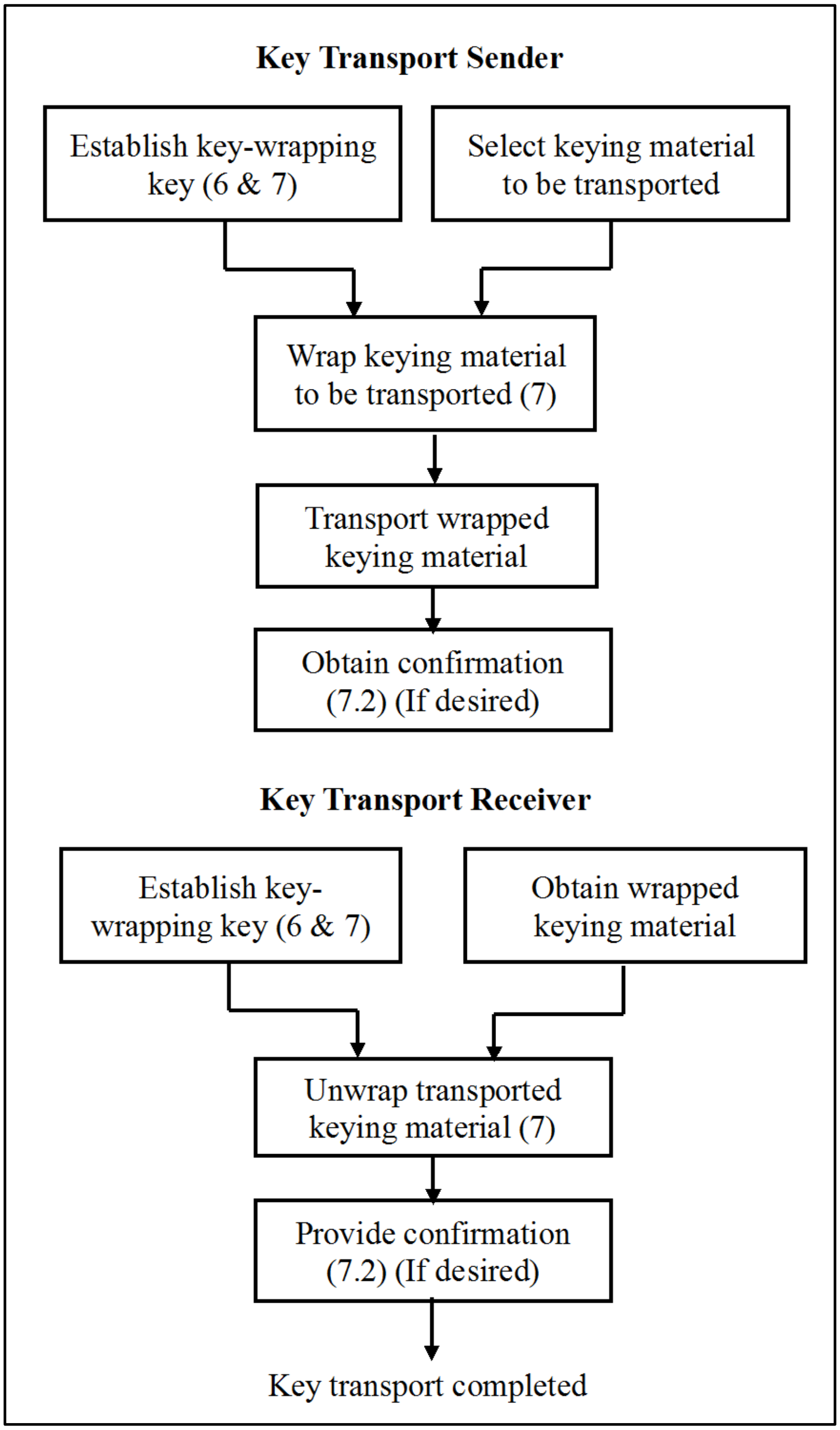

Figure 3: Key-transport process

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 and FIPS 202 specify **approved** hash functions.

5.2 Message Authentication Code (MAC) Algorithm

A Message Authentication Code (MAC) algorithm defines a family of cryptographic functions that is parameterized by a symmetric key. It is computationally infeasible to determine the MAC of a (newly formed) *MacData* value without knowledge of the *MacKey* value (even if one has seen the MACs corresponding to other *MacData* values that were computed using that same *MacKey* value).

The input to a MAC algorithm includes a symmetric key, called *MacKey* and a binary data string called *MacData* that serves as the "message." That is, a MAC computation is represented as MAC(*MacKey*, *MacData*). In this Recommendation, a MAC algorithm is used if key confirmation is performed during key establishment (see Section 5.9); a (possibly different) MAC algorithm may be used for the required key-derivation process (see SP 800-56C).

Key confirmation requires the use of an **approved** MAC algorithm, i.e., HMAC, AES-CMAC or KMAC. HMAC is specified in FIPS 198 and requires the use of an **approved** hash function. AES-CMAC is specified in SP 800-38B for the AES block cipher algorithm specified in FIPS 197. KMAC is specified in SP 800-185.

When used for key confirmation, an entity is required to compute a MAC tag on received or derived data using a MAC algorithm with 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.

If a MAC algorithm is employed in key derivation, an **approved** MAC algorithm **shall** be selected and used in accordance with SP 800-56C.

5.2.1 MAC Tag Computation for Key Confirmation

Key confirmation can be performed as part of a key-agreement scheme, following key transport or during both processes.

The computation of a MAC tag (denoted *MacTag*) is represented as follows:

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

To compute a *MacTag*:

1. The agreed-upon MAC algorithm (see Section 5.2) is used with *MacKey* to compute the MAC of *MacData*, where *MacKey* is a symmetric key, and *MacData* represents the input "message" data. The minimum length of *MacKey* is specified in Table 6 and Table 7 of Section 5.9.3.

   MacKey is obtained from the *DerivedKeyingMaterial* (when a key-agreement scheme employs key confirmation), as specified in Section 5.9.1.1, or obtained from the transported keying material, *KM* (when a key-transport scheme employs key confirmation), as specified in Section 7.2.

   The output of the MAC algorithm is a bit string whose length is *MacOutputLen* bits."

2. Those *MacOutputLen* bits are input to the truncation function $T_{MacTagLen}$, which returns the leftmost (i.e., initial) *MacTagLen* bits to be used as the value of *MacTag*. *MacTagLen* **shall** be less than or equal to *MacOutputLen*. (When *MacTagLen* equals *MacOutputLen*, $T_{MacTagLen}$ acts as the identity function.) The minimum value for *MacTagLen* is 64, as specified in Section 5.9.3.

5.2.2 MAC Tag Verification for Key Confirmation

To verify a received *MacTag* (i.e., received during key confirmation), a new MAC tag, *MacTag′* is computed using the values of *MacKey*, *MacTagLen*, 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*, *MacTagLen*, 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 at an appropriate security strength using an **approved** random bit generator (see the SP 800-90 series of publications).

5.4 Nonce

A nonce is a time-varying value that has an acceptably small chance of repeating (where the meaning of "acceptably small" 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 a party 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 bit string 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 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 a nonce is required for key-agreement and/or key-confirmation purposes as specified in this Recommendation, it **should** be a random nonce. The security strength supported by the instantiation of the random bit generator and the bit length of the random bit string **shall** be equal to or greater than the targeted security strength of the key-agreement scheme in which it is used. However, the bit length of the random bit string **should** be (at least) twice the targeted security strength.

For details concerning the security strength supported by an instantiation of a random bit generator, see SP 800-90.

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 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 set of domain parameters.

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 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 Selection/Generation

5.5.1.1 FFC Domain Parameter Selection/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 ($p, q, g\{, SEED, counter\}$), where p is the (odd) prime field size, q is an (odd) prime divisor of $p - 1$, and g is a generator of the cyclic subgroup of $GF(p)^*$ of order q. The optional parameters, *SEED* and *counter*, are described below.

Two classes of domain parameters are **approved** for FFC key agreement: the class of "safe" domain parameters that are associated with **approved** safe-prime groups, and the class of "FIPS 186-type" domain parameters that conform to one of the FIPS 186-type parameter-size sets that are listed in Table 1.

The safe-prime groups **approved** for use by U.S. Government applications are listed in Appendix E. The associated domain parameters have the form (p, $q = (p - 1)/2$, $g = 2$) for specific choices of p. (There are no *SEED* or *counter* values required for these groups as there are for the FIPS 186-type groups; see below.) Appendix E specifies the security strengths that can be supported by the **approved** safe-prime groups.

The generation of FIPS 186-type domain parameters conforming to parameter-size set FB or FC from Table 1 **shall** be performed as specified in FIPS 186. The resulting domain parameters are of the form ($p, q, g\{, SEED, counter\}$), where *SEED* and *counter* are parameters used in an **approved** process for generating and validating p, q, and possibly g (depending on the method of generation). The party that generated the domain parameters **should** retain *SEED* and *counter* and make them available upon request for domain-parameter validation.

When the targeted security strength for key establishment is greater than 112 bits, an **approved** safe-prime group capable of supporting that security strength **shall** be used. When the targeted security strength is 112 bits, an **approved** safe-prime group **should** be used. The use of FIPS 186-type domain parameters **should** only be used when the targeted security strength is 112 bits for backward compatibility with existing applications that cannot be upgraded to use the **approved** safe-prime groups.

Table 1: FIPS 186-type FFC parameter-size sets[1]	**FB**	**FC**
Targeted security strength (in bits)	112	112
Bit length of field size p (i.e., len(p))	2048	2048
Bit length of subgroup order q (i.e., len(q))	224	256

In the binary representation of each of the odd primes p and q, both the leftmost bit and the rightmost bit **shall** be a 1 (i.e., no padding is permitted to artificially increase the bit lengths of their representations).

The (safe or FIPS 186-type) domain parameters used for FFC key agreement **shall** be selected in accordance with the targeted security strength of the relying key-establishment scheme. SP 800-57 provides guidance on determining security strengths that are appropriate for various applications.

5.5.1.2 ECC Domain-Parameter Selection

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. For the purposes of this Recommendation, an elliptic curve defined over GF(q) is assumed to be defined by 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 $GF(q)$. In such an equation, the indicated arithmetic is performed in $GF(q)$. (See [SECG] or Annexes A.2, G.1, and G.2 of ANS X9.62 for further information concerning arithmetic in finite fields.) 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. The set of elliptic curve points forms a group, given an appropriate binary operation "+" (elliptic-curve addition, as defined by the well-known secant-and-tangent rules) and the introduction of a special "point at infinity" to serve as "Ø" (the additive identity element). (See [SECG] or ANS X9.62 for the details of elliptic-curve group operations.)

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

$$\underbrace{G + G + \ldots + G,}$$

[1] An additional parameter-size set (FA) that provides a maximum security strength of 80 bits is **no longer approved** for use (see SP 800-57 and SP 800-131A).

d terms

where "+" is the elliptic-curve addition operation, then the elements of the cyclic subgroup can be represented as {Ø, *G*, 2*G*,…, (*n* – 1)*G*}. Note that *nG* = Ø. The full elliptic-curve group 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 used to provide additional information (as specified in ANS X9.63 or SECG) concerning 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. When $q = 2^m$, the elements of $GF(2^m)$ are represented by bit strings of length *m*, with each bit indicating the coefficient (0 or 1) of a specific element of a particular basis for $GF(2^m)$ viewed as a vector space over *GF*(2). *FR* is *Null* if $q = 2^m$ and the representation of field elements corresponds to a Gaussian normal basis for $GF(2^m)$ (as specified in Annex A.2.3.3 of ANS X9.62, and further described in Annexes G.2.4, G.2.5, and H.1 of that document). If $q = 2^m$, and the representation of field elements corresponds to a polynomial basis (as specified in [SECG] or Annex A.2.3.2 of ANS X9.62, and further described in Annexes G.2.2, G.2.3, H.2, and H.3 of that document), then *FR* specifies 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 an **approved** process for generating and validating *a*, *b*, and possibly *G* (depending on the method of generation).

The ECC domain parameters for U.S. Government applications **shall** be selected from the recommended elliptic-curve domain parameters in SP 800-186[2]. The names of these curves are also listed in Appendix E, along with the security strengths that can be supported by each curve. The curves to be used for ECC key agreement **shall** be selected in accordance with the targeted security strength of the relying key-establishment scheme. SP 800-57 provides guidance on determining the security-strength requirements that are appropriate for various applications.

5.5.2 Assurances of Domain-Parameter Validity

Secure key establishment depends on the arithmetic validity of the domain parameters used by the parties. Therefore, each party **shall** have assurance of the validity of candidate domain parameters before they are used for key establishment. Each party **shall** obtain assurance that the candidate domain parameters are valid in one of the following ways:

1. The domain parameters correspond to a specifically **approved** group:

[2] The recommended elliptic curves now listed in FIPS 186 will be moved to SP 800-186. Until SP 800-186 is published, the recommended elliptic curves should be taken from FIPS 186-4.

   a. For FFC: An **approved** safe-prime group, as listed in Appendix E.

   b. For ECC: An elliptic-curve group **approved** for use by the key-establishment schemes specified in this Recommendation, as listed in SP 800-186.

2. For FFC domain parameters that conform to a FIPS 186-type parameter-size set (see Table 1):

   a. The party has generated the domain parameters using a method specified in FIPS 186, and/or

   b. The party has performed an explicit domain-parameter validation as specified in FIPS 186, using the provided *SEED* and *counter* values.

   (Method b can be used by the party that generated the FFC domain parameters to obtain renewed assurance of their validity, as necessary.)

3. A trusted third party (for example, a CA) has obtained assurance that the domain parameters are valid in accordance with one of the methods above, and has communicated that fact through a trusted channel.

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 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 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.

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

Each FFC static and ephemeral key pair **shall** be generated using an **approved** method (see Section 5.6.1.1.3 or 5.6.1.1.4) and the selected valid domain parameters (*p, q, g*{, *SEED, counter*}).

5.6.1.1.1 Using the Approved Safe-Prime Groups

When the domain parameters (p, $q = (p – 1)/2$, $g = 2$) correspond to an **approved** safe-prime group (named in Appendix E), private keys are integers in [1, q – 1] whose binary representations require no more than N bits, for an appropriate choice of N, and the corresponding public keys are in [2, p – 2]. For the key-pair generation methods in Sections 5.6.1.1.3 and 5.6.1.1.4, the value of the input parameter s **shall** be the largest security strength that can be supported by the named safe-prime group, and the value for the input parameter N (the requested maximum bit length of the private key) **shall** satisfy the inequalities $2s \leq N \leq \text{len}(q)$. The generated key pairs **shall** be used only for key-establishment purposes (see Sections 6 and 7 for the **approved** key-establishment schemes).

5.6.1.1.2 Using the FIPS 186-Type FFC Parameter-size Sets

When the domain parameters (*p, q, g*{, *SEED, counter*}) conform to a FIPS 186-type FFC parameter-size set (see Table 1), private keys are generated in [1, q – 1], and the corresponding public keys are in [2, p – 2]. For the key-pair generation methods in Sections 5.6.1.1.3 and 5.6.1.1.4, the value used for the input parameter N **shall** be len(q), i.e., the bit length of the domain parameter q, and the value used for the input parameter s **shall** be 112, which is the security strength that can be supported by the FIPS 186-type FFC parameter-size set that was used to generate the domain parameters (see Table 1). The generated key pairs **shall** be used only for key-establishment purposes (see Sections 6 and 7 for the **approved** key-establishment schemes), with the possible exception discussed in item 5 of Section 5.6.3.2.

5.6.1.1.3 Key-Pair Generation Using Extra Random Bits

In this method, 64 more bits are requested from the random bit generator (RBG) than are needed for the private key so that bias produced by the mod function in process step 5 is negligible.

The following process or its equivalent may be used to generate an FFC key pair.

Input:

1. (*p, q, g*) The FFC domain parameters used by this process. *p*, *q* and *g* **shall** either be provided as integers during input, or **shall** be converted to integers prior to use.
2. *N* The (maximum) bit length of the private key to be generated.
3. *s* The maximum security strength to be supported by the key pair.

Output:

1. *status* The status returned from the key-pair generation process. The status will indicate **SUCCESS** or an **ERROR**.

2. (x, y) The generated private and public keys. If an error is encountered during the generation process, invalid values for x and y **should** be returned, as represented by *Invalid_x* and *Invalid_y* in the following specification; for example, both *Invalid_x* and *Invalid_y* could be 0. Otherwise, x and y are returned as integers. The generated private key x is in $[1, \min(2^N - 1, q - 1)]$, and the public key y is in the interval $[2, p - 2]$.

Process:

1. If s is not the maximum security strength that can be supported by (p, q, g), then return an **ERROR** indication as the *status* and (*Invalid_x*, *Invalid_y*) as the key pair; then exit the process without performing the remaining steps.
2. If $((N < 2s)$ or $(N > \text{len}(q)))$, then return an **ERROR** indication as the *status* and (*Invalid_x*, *Invalid_y*) as the key pair; then exit the process without performing the remaining steps.
3. Obtain a string of $N + 64$ *returned_bits* using an RBG with a security strength of s bits or more (see Section 5 in SP 800-133). If an **ERROR** indication is returned, then return an **ERROR** indication as the *status* and (*Invalid_x*, *Invalid_y*) as the key pair; then exit the process without performing the remaining steps.
4. Convert *returned_bits* to the (non-negative) integer c in the interval $[0, 2^{(N+64)} - 1]$ (see Appendix C.4).
5. Set $M = \min(2^N, q)$, the minimum of 2^N and q.
6. Set $x = (c \bmod (M - 1)) + 1$.
7. Set $y = g^x \bmod p$.
8. Return **SUCCESS** as the *status* and (x, y) as the key pair.

Output: **SUCCESS** and (x, y), or

an **ERROR** indication and (*Invalid_x*, *Invalid_y*).

5.6.1.1.4 Key-Pair Generation by Testing Candidates

In this method, a random number is obtained and tested to determine whether it will produce a value for the private key in the correct interval. If the private key would be outside the interval, then another random number is obtained (i.e., the process is iterated until an acceptable value for the private key is obtained).

The following process or its equivalent may be used to generate an FFC key pair.

Input:

1. (p, q, g) The FFC domain parameters used by for this process. p, q and g **shall** either be provided as integers during input, or **shall** be converted to integers prior to use.
2. N The (maximum) bit length of the private key to be generated.
3. s The maximum security strength to be supported by the key pair.

Output:

1. *status* The status returned from the key-pair generation process. The status will indicate **SUCCESS** or an **ERROR**.

2. (x, y) The generated private and public keys. If an error is encountered during the generation process, invalid values for x and y **should** be returned, as represented by *Invalid_x* and *Invalid_y* in the following specification; for example, both *Invalid_x* and *Invalid_y* could be 0. Otherwise, x and y are returned as integers. The generated private key x is in $[1, \min(2^N - 1, q - 1)]$, and the public key y is in the interval $[2, p - 2]$.

Process:

1. If s is not the maximum security strength that can be supported by (p, q, g), then return an **ERROR** indication as the *status* and (*Invalid_x*, *Invalid_y*) as the key pair; then exit the process without performing the remaining steps.

2. If $((N < 2s)$ or $(N > \text{len}(q)))$, then return an **ERROR** indication as the *status* and (*Invalid_x*, *Invalid_y*) as the key pair; then exit the process without performing the remaining steps.

3. Obtain a string of N *returned_bits* using an RBG with a security strength of s bits or more (see Section 5 of SP 800-133). If an **ERROR** indication is returned, then return an **ERROR** indication as the *status* and (*Invalid_x*, *Invalid_y*) as the key pair; then exit the process without performing the remaining steps.

4. Convert *returned_bits* to the (non-negative) integer c in the interval $[0, 2^N - 1]$ (see Appendix C.4).

5. Set $M = \min(2^N, q)$, the minimum of 2^N and q.

6. If $(c > M - 2)$, then go to step 3.

7. $x = c + 1$.

8. $y = g^x \bmod p$.

9. Return **SUCCESS** as the *status* and (x, y) as the key pair.

Output: SUCCESS and (x, y), or

an **ERROR** indication and (*Invalid_x, Invalid_y*).

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 Section 5.6.1.2.1 and 5.6.1.2.2) and domain parameters that have been selected in accordance with Section 5.5.1.2. For the key-pair generation methods in Sections 5.6.1.2.1 and 5.6.1.2.2, the value of the input parameter s **shall** be the maximum security strength that can be supported by the corresponding elliptic-curve group, as specified in Appendix E.

Given valid domain parameters, each valid private key *d* is an integer that is randomly selected in the interval [1, *n*–1]. Whether static or ephemeral, each valid public key *Q* is related to the corresponding (valid) private key *d* by the following formula: $Q = (x_Q, y_Q) = dG$.

5.6.1.2.1 Key Pair Generation Using Extra Random Bits

In this method, 64 more bits are requested from the RBG than are needed for *d* so that bias produced by the mod function in step 6 is negligible.

The following process or its equivalent may be used to generate an ECC key pair.

Input:

1. (*q, FR, a, b* {, *domain_parameter_seed*}, *G, n, h*)

   The ECC domain parameters that are used for this process. *n* is a prime number, and *G* is a point on the elliptic curve (with additive order *n*).

2. *s* The maximum security strength to be supported by the key pair.

Output:

1. *status* The status returned from the key-pair generation procedure. The status will indicate **SUCCESS** or an **ERROR**.

2. (*d*, *Q*) The generated private and public keys. If an error is encountered during the generation process, invalid values for *d* and *Q* **should** be returned, as represented by *Invalid_d* and *Invalid_Q* in the following specification; for example, *Invalid_d* and *Invalid_Q* could be a point that is not on the elliptic curve defined by the domain parameters. The private key *d* is an integer in the interval [1, *n*–1], and *Q* is an elliptic curve point.

Process:

1. If the domain parameters are not **approved**, then return an **ERROR** indication as the *status* and (*Invalid_d*, *Invalid_Q*) as the key pair; then exit the process without performing the remaining steps.

2. If *s* is not the maximum security strength that can be supported by the domain parameters, then return an **ERROR** indication as the *status* and (*Invalid_d*, *Invalid_Q*) as the key pair; then exit the process without performing the remaining steps.

3. $L = \text{len}(n) + 64$.

4. Obtain a string of *L* *returned_bits* using an RBG with a security strength of *s* bits or more (see Section 5 in SP 800-133). If an **ERROR** indication is returned, then return an **ERROR** indication as the *status* and (*Invalid_d*, *Invalid_Q*) as the key pair; then exit the process without performing the remaining steps.

5. Convert *returned_bits* to the (non-negative) integer *c* in the interval $[0, 2^L - 1]$ (see Appendix C.4).

6. $d = (c \bmod (n-1)) + 1$.

7. $Q = dG$.
8. Return **SUCCESS** as the *status* and (*d*, *Q)* as the key pair.

Output: SUCCESS and (*d*, *Q*), or

an **ERROR** indication and (*Invalid_d, Invalid_Q*).

5.6.1.2.2 Key Pair Generation by Testing Candidates

In this method, a random number is obtained and tested to determine whether or not it will produce a value of *d* in the correct interval. If *d* would be outside the interval, another random number is obtained (i.e., the process is iterated until an acceptable value of *d* is obtained.

The following process or its equivalent may be used to generate an ECC key pair.

Input:

1. (*q, FR, a, b* {, *domain_parameter_seed*}*, G, n, h*)

   The ECC domain parameters that are used for this process. *n* is a prime number, and *G* is a point on the elliptic curve (with the additive order *n*).

2. *s* The maximum security strength to be supported by the key pair.

Output:

1. *status* The status returned from the key pair generation procedure. The status will indicate **SUCCESS** or an **ERROR**.

2. (*d*, *Q*) The generated private and public keys. If an error is encountered during the generation process, invalid values for *d* and *Q* **should** be returned, as represented by *Invalid_d* and *Invalid_Q* in the following specification; for example, *Invalid_d* and *Invalid_Q* could be a point that is not on the elliptic curve defined by the domain parameters. *d* is an integer in the interval [1, *n*–1], and *Q* is an elliptic curve point.

Process:

1. If the domain parameters are not **approved**, then return an **ERROR** indication as the *status* and (*Invalid_d*, *Invalid_Q*) as the key pair; then exit the process without performing the remaining steps.
2. If *s* is not the maximum security strength that can be supported by the domain parameters, then return an **ERROR** indication as the *status* and (*Invalid_d*, *Invalid_Q*) as the key pair; then exit the process without performing the remaining steps.
3. $L = \text{len}(n)$.
4. Obtain a string of *L returned_bits* using an RBG with a security strength of *s* bits or more (see Section 5 in SP 800-133). If an **ERROR** indication is returned, then return an **ERROR** indication as the *status* and (*Invalid_d*, *Invalid_Q*) as the key pair; then exit the process without performing the remaining steps.

5. Convert *returned_bits* to the (non-negative) integer c in the interval $[0, 2^L - 1]$ (see Appendix C.4).
6. If ($c > n-2$), then go to step 4.
7. $d = c + 1$.
8. $Q = dG$.
9. Return **SUCCESS** as the *status* and (d, Q) as the key pair.

Output: SUCCESS and (d, Q), or

an **ERROR** indication and (*Invalid_d*, *Invalid_Q*).

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 3 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 interval, 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 2 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 2: Initial assurances required by the key-pair owner

	Types of assurance			
Key-pair type	**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, methods 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 3 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 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 3: Optional renewal of assurances by the key-pair owner

	Types of assurance			
Key-pair type	**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 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. For both a static and ephemeral key pair: The owner generates the key pair as specified in Section 5.6, or

b. For a static key pair (only): A trusted third party (TTP) (trusted by the owner and any recipient of the public key) generates the 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 interval, which depends on the type of domain parameters that are used to generate key pairs.

- When FFC domain parameters (*p*, *q*, *g*{, *SEED*, *counter*}) are used that conform to a FIPS 186-type FFC parameter-size set from Table 1, private keys are in the interval $[1, q - 1]$.
- When an **approved** safe-prime group is used (see Section 5.5.1.1), and the corresponding FFC domain parameters are (p, $q = (p - 1)/2$, $g = 2$), the private keys are in the interval $[1, M - 1]$, where $M = \min(2^N, q)$, and N is the agreed-upon (maximum) bit length, satisfying $2s \le N \le \text{len}(q)$, where s is the maximum security strength that can be supported by the safe-prime group, as specified in Appendix E.
- When an **approved** elliptic-curve group is used, and the corresponding ECC domain parameters are (*q*, *FR*, *a*, *b*{, *SEED*}, *G*, *n*, *h*), the private keys are in the interval $[1, n - 1]$.

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 interval by using one of the following methods:

a. For both a static and ephemeral key pair: The owner generates the key pair as specified in Section 5.6.1, or

b. For a static key pair (only): After receiving a 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 interval. (While an entity can accept ownership of a static key pair that was generated by a TTP, an ephemeral key pair **shall** only be generated by its owner.)

To renew this assurance for a static key pair (if desired), the owner **shall** perform a separate check to determine that the private key is in the correct interval 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 public key has the expected representation for a non-identity element of the correct cryptographic subgroup, as determined by the domain parameters, using one of the following methods:

a. For either a static key pair or an ephemeral key pair: 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 for FFC and 5.6.2.3.3 for ECC); or

b. For either a static key pair or an ephemeral key pair: 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 5.6.2.3.3) (either the owner or a TTP could have generated a static key pair; only the owner can generate an ephemeral key pair); or

c. For a static key pair (only): A trusted third party (TTP) (trusted by the owner) performs a full public-key validation (see Sections 5.6.2.3.1 and 5.6.2.3.3) 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 **shall** either 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 perform a full public-key validation as a separate process, following its key-pair generation process.

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 for FFC and 5.6.2.3.3 for ECC). 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. For either a static key pair or an ephemeral key pair: The owner generates the key pair as specified in Section 5.6.1, or

b. For a static key pair (only): Subsequent to the generation of a static key pair by the owner or a trusted third party as specified in Section 5.6.1, the owner performs one of the following consistency tests (as appropriate for the FCC or ECC domain parameters used during the generation process).

- For an FFC key pair (x, y): Use the private key, x, along with the generator g and prime modulus p included in the domain parameters associated with the key pair to compute g^x mod p. Compare the result to the public key, y. If g^x mod p is not equal to y, then the pair-wise consistency test fails.

- For an ECC key pair (d, Q): Use the private key, d, along with the generator G and other domain parameters associated with the key pair, to compute dG (according to the rules of elliptic-curve arithmetic). Compare the result to the public key, Q. If dG is not equal to Q, then the pair-wise consistency test fails.

The static public key **shall** be successfully recomputed from the private key and the domain parameters to obtain assurance (via method b) that the private and public keys are consistent. If this pair-wise consistency test fails, the tested key pair **shall not** be used.

To renew assurance of pair-wise consistency for a static key pair (if desired), 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. For either a static key pair or an ephemeral key pair: The owner generates the key pair as specified in Section 5.6.1, or

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

To renew this assurance for a static private key (if desired), the appropriate 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

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 assurance of public-key validity and/or private-key possession as required below:

- 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. Recipients of static public keys are required to obtain this assurance (see Section 5.6.2.2.1). Recipients of ephemeral public keys are also required to obtain this assurance.

- 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. Recipients of static public keys are required to obtain this assurance (see Section 5.6.2.2.3). Recipients of ephemeral public keys are encouraged (but not required) to obtain this assurance; (optional) methods for obtaining this assurance are discussed in Section 5.6.2.2.4.

Table 4 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 4: Assurances required by a public-key recipient

	Type of assurance	
Key-pair type	**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.

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 indicated methods for obtaining the required (and/or desired) assurances meet their security requirements. The application tasked with performing key establishment on behalf of the recipient **should** determine whether to proceed with a key-establishment transaction, based upon the perceived adequacy of the method(s) used to obtain the assurances described above.

Once the necessary steps have been taken to provide the recipient of a static public key with assurance of its validity, the assurance obtained by the recipient may endure for a protracted period without the need to reconfirm the validity of that public key. The same may be true of assurance provided to the recipient that the owner of the static public key possesses the corresponding static private key. This could be the case, for example, when the source of the assurance is a trusted CA whose (valid) signature on a certificate containing the static public key indicates to the recipient that the arithmetic validity of the static public key has been confirmed by the CA and that the owner's possession of the corresponding static private key has been established to the CA's satisfaction. Alternatively, a party could maintain a record (i.e., an integrity-protected record) of previously received static public keys whose validity was confirmed and/or whose owners have provided assurance of private-key possession.

On the other hand, the recipient of a static public key may choose to obtain renewed assurance of its validity and/or choose to obtain renewed assurance that the owner of the static public key (i.e., the other party) possesses the corresponding static private key. Deciding how often (if at all) to seek renewed assurance is a determination that **should** be made by the recipient (or an agent trusted to act on the recipient's behalf), based on the recipient's security needs.

Renewed assurance of the validity of a received ephemeral public key and renewed assurance that the other party is in possession of the corresponding ephemeral private key are not addressed in this Recommendation, since ephemeral key pairs are short-lived.

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 for FFC and 5.6.2.3.3 for ECC).
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 5.6.2.3.3). This TTP could, for example, be a binding authority, such as a CA (see Section 4.1).

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 its validity by using one of the following methods:

1. When an **approved** FFC safe-prime group or an **approved** elliptic curve group is used by the key-establishment scheme:
   - The recipient performs a successful partial public-key validation on the received public key (see Section 5.6.2.3.2 for FFC domain parameters and Section 5.6.2.3.4 for ECC domain parameters); or
   - The recipient performs a successful full public-key validation on the received public key (see Section 5.6.2.3.1 for FFC domain parameters and Section 5.6.2.3.3 for ECC domain parameters).

   (As part of the proper implementation of this Recommendation, system users and/or agents trusted to act on their behalf **should** determine whether a partial validation of ephemeral public keys is sufficient to meet their security requirements. If it is determined that partial public-key validation is insufficient, then full public-key validation **shall** be performed.)
2. When FIPS 186-type FFC domain parameters are used in the key-establishment scheme: The recipient performs a successful full public-key validation on the received public key (see Section 5.6.2.3.1 for FFC domain parameters).

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) 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 (unless the assurance of public-key validity and assurance of private-key possession are obtained simultaneously from a trusted third party).

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 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.

A binding authority can be used to bind the key-pair owner's identifier to his static public key. In this case, 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 (only if using the FIPS 186-type domain parameters or the **approved** ECC domain parameters) by using an **approved** alternative (see SP 800-57, Sections 5.2 and 8.1.5.1.1.2). Note that the use of the signature-based alternative described in SP 800-57 is **not approved** for the safe-prime domain parameters.

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 (however, see the discussion in Section 5.6.2.2.3 above).

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 given 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 the following conditions are met during a key-agreement transaction that incorporates key confirmation as specified in this Recommendation, then while 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 ownership of the received static public key and current knowledge of the corresponding static private key.)

There are several key-agreement schemes specified in this Recommendation that can be used while satisfying both of the conditions above. 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 Section 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 an application. Assurance of the validity of the ephemeral public key **shall** be obtained prior to obtaining assurance of possession of the private key.

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 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 of that ephemeral public key) 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 static and ephemeral public keys contributes its own (distinct) ephemeral public key to the key-agreement process, one that is intended to be arithmetically combined with the private key corresponding to the received ephemeral public key in computations performed by the claimed owner of the received static and ephemeral public keys. (If an appropriate key-agreement scheme is employed, the claimed owner of the received public keys 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 static and ephemeral public keys is also a key confirmation recipient, with the claimed owner of the received public keys serving as the key-confirmation provider. (By successfully providing key confirmation, the claimed owner of the received public keys can demonstrate that he 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. 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 Section 6.1.1.2 and Section 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

Public-key validation refers to the process of checking the arithmetic properties of a candidate public key. Both full and partial validation routines are provided for public keys that are associated with either FFC or ECC domain parameters. Public-key validation does not require knowledge of the associated private key and so may be done at any time by anyone. However, these routines assume a prior validation of the domain parameters

5.6.2.3.1 FFC Full Public-Key Validation Routine

FFC full public-key validation refers to the process of checking the arithmetic properties of a candidate FFC public key to ensure that it has the expected representation and is in the correct subgroup of the multiplicative group of the finite field specified by the associated FFC domain parameters.

This routine **shall** be used when assurance of full public-key validity is required (or desired) for a static or ephemeral FFC public key.

Input:

1. (p, q, g{, *SEED*, *counter*}): A valid set of FFC domain parameters, and
2. y: A candidate FFC public key.

Process:

1. Verify that $2 \le y \le p - 2$.

   Success at this stage ensures that y has the expected representation for a nonzero field element (i.e., an integer in the interval [1, p – 1]) and that y is in the proper range for a properly generated public key.

2. Verify[3] that $1 = y^q \bmod p$.

   Success at this stage ensures that y has the correct order and thus, is a non-identity element in the correct subgroup of $GF(p)^*$.

Output: If any of the above verifications fail, immediately output an error indicator and exit without further processing. Otherwise, output an indication of successful validation.

5.6.2.3.2 FFC Partial Public-Key Validation Routine

FFC partial public-key validation refers to the process of performing only the first step of a full public-key validation, omitting the check that determines whether the candidate FFC public key is in the correct subgroup.

This routine **shall** only be used with ephemeral FFC public keys generated using the **approved** safe-prime groups when assurance of the partial validity of such keys is to be obtained as specified in Section 5.6.2.2.2.

Input:

1. (p, $q = (p - 1)/2$, $g = 2$) A valid set of "safe" FFC domain parameters corresponding to a safe-prime group (see Section 5.5.1.1), and
2. y: A candidate FFC public key.

Process:

[3] When the FFC domain parameters correspond to a safe-prime group, $1 = y^q \bmod p$ if and only if y is a (nonzero) quadratic residue modulo p, which can be verified by computing the value of the Legendre symbol of y with respect to p.

Verify that $2 \le y \le p - 2$.

Success at this stage ensures that y has the expected representation for a nonzero field element (i.e., an integer in the interval $[1, p - 1]$) and that y is in the proper range for a properly generated public key.

Output: If the above verification fails, output an error indicator. Otherwise, output an indication of successful validation.

5.6.2.3.3 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 expected representation for a non-identity element of the correct subgroup of the appropriate elliptic-curve group, as specified by the associated ECC domain parameters.

This routine **shall** be used when assurance of full public-key validity is required (or desired) for a static or ephemeral ECC public key.

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 identity element $\varnothing$.

   Success at this stage ensures that Q is not the identity element of the elliptic-curve group (which would never be the value of a properly generated public key).

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 $q = 2^m$.

   Success at this stage ensures that each coordinate of the public key has the expected representation for an element in the underlying field, $GF(q)$.

3. Verify that Q is on the curve. In particular,

   - If q is an odd prime p, verify that $(y_Q)^2 = ((x_Q)^3 + ax_Q + b) \bmod p$.
   - 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.

   Success at this stage ensures that the public key is a point on the correct elliptic curve.

4. Compute nQ (using elliptic curve arithmetic), and verify that $nQ = \varnothing$.

   Success at this stage ensures that the public key has the correct order. Along with the successful verifications in the previous steps, this step ensures that the public key is in the correct elliptic-curve subgroup and is not the identity element.

Output: If any of the above verifications fail, immediately output an error indicator and exit without further processing. Otherwise, output an indication of successful validation.

5.6.2.3.4 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 has the expected representation for a non-identity element of the correct elliptic-curve group, as specified by the associated ECC domain parameters. ECC partial public-key validation omits the validation of subgroup membership[4], and therefore, is usually faster than ECC full public-key validation.

This routine **shall** only be used when assurance of partial public-key validity is acceptable for an ephemeral ECC public key.

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 identity element $Ø$.

   Success at this stage ensures that Q is not the identity element of the elliptic-curve group (which would never be the value of a properly generated public key).

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 $q = 2^m$.

   Success at this stage ensures that each coordinate of the public key has the expected representation for an element in the underlying field, $GF(q)$.

3. Verify that Q is on the curve. In particular,

   - If q is an odd prime p, verify that $(y_Q)^2 = ((x_Q)^3 + ax_Q + b) \bmod p$.
   - 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.

   Together with the successful verifications in the previous steps, success at this stage ensures that the public key is a (finite) point on the correct elliptic curve.

   (Note: Since its order is not verified, there is no check that the public key is in the correct elliptic curve subgroup. The cofactor multiplication employed by the ECC primitives used to compute a shared secret is intended to compensate for this omission.)

Output: If any of the above verifications fail, immediately output an error indicator and exit without further processing. Otherwise, output an indication of validation success.

---

[4] In this Recommendation, co-factor multiplication is included in the ECC primitives for Diffie-Hellman and MQV, which forces the computed group element into the appropriate subgroup.

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 ECC key pairs (see SP 800-57):

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 for static public keys by using public-key certificates that have been signed by a Certification Authority (CA). Ephemeral public keys may be protected during communication using digital signatures or other protocol-specific methods.

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, using 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; key-usage restrictions could be by a CA when generating certificates) with the following possible exception for ECC and FIPS 186-type FFC domain parameters: 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 on Key Usage for further information. A key-establishment key pair generated using safe-prime domain parameters **shall not** ever be used for the generation of a digital signature.

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). In either case, after its use, an ephemeral private key **shall** be destroyed as soon as possible. Until the private key is destroyed, its confidentiality **shall** be protected. An ephemeral private key **shall not** be backed up or archived.

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 the technique used by the owner to generate the ephemeral key pair. See Section 5.6.2.1 for further details.

4. A recipient of an ephemeral public key **shall** have assurance of the full or partial validity of the public key as specified in 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, 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 (see Section 5.7.1.1): 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 (called the Modified Diffie-Hellman primitive in ANS X9.63; see Section 5.7.1.2 below): 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 (see Section 5.7.2.1): 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 (see Section 5.7.2.3): 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*, *counter*}), 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*, *counter*}): 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 (($z \leq 1$) OR ($z = p - 1$)), destroy all intermediate values used in the attempted computation of *Z* (including *z*), then output an error indicator, and exit this process without further processing.
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 $P = \varnothing$, destroy all intermediate values used in the attempted computation of *P*, then output an error indicator, and exit this process without further processing.
3. Else, set $z = x_P$, where x_P is the *x*-coordinate of *P*, and convert *z* to *Z*, using the field-element-to-byte string conversion routine defined in Appendix C.2.
4. Destroy the results of all intermediate calculations used in the computation of *Z* (including *P* and *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, counter*}): 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 (($z \leq 1$) OR ($z = p - 1$)), destroy all intermediate values (including T_A, S_A, and T_B) used in the attempted computation of z, then output an error indicator, and exit this process without further processing.
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 T_A, S_A, T_B, and 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.

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

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 uses the MQV primitive with 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 is party V's static public key (y_B).

Party V uses his/her static private key for the second private key; that is, when party V uses the algorithm in Section 5.7.2.1, item 4 of the input list is 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 identity element $\emptyset$).

Process:

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

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

3. Calculate the associate value function

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

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

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:

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

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 $P = \emptyset$, destroy all intermediate values used in the attempted computation of P, then output an error indicator, and exit this process without further processing.
4. Else, set $z = x_P$, where x_P is the x-coordinate of P, and convert z to Z, using the field-element-to-byte string conversion routine defined in Appendix C.2."
5. Destroy the results of all intermediate calculations used in the computation of Z (including P and 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 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).

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

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, item 6 of the input list is 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 is V's static private key $d_{s,A}$, and item 5 is his static public key ($Q_{s,A}$).

5.8 Key-Derivation Methods for Key-Agreement Schemes

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. The shared secret **shall** be used only by an **approved** key-derivation method and **shall not** be used for any other purpose.

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*, along with other information. The derived keying material **shall** be computed in its entirety before outputting any portion of it, and (each copy of) *Z* **shall** be treated as a critical security parameter and 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 key-derivation key that will be used to generate additional keying material (possibly using a different process – see SP 800-108). The derived keying material **shall not** be used as a key stream for a stream cipher. Non-secret keying material (such as a non-secret initialization vector) **shall not** be generated using a key-derivation method that includes the shared secret, *Z*, as input (this restriction applies to all one-step and two-step key-derivation methods).

5.8.1 Performing the Key Derivation

Approved methods for key derivation from a shared secret are specified in SP 800-56C. These methods can be accessed using the following call:

$$\text{KDM}(Z, OtherInput),$$

where

1. *Z* is a byte string that represents the shared secret,
2. *OtherInput* consists of additional input information that may be required by a given key-derivation method, for example:
   - *L* − an integer that indicates the length (in bits) of the secret keying material to be derived.
   - *salt* − a byte string.
   - *IV* – a bit string used as an initialization value.
   - *FixedInfo* – a bit sting of context-specific data (see Section 5.8.2).

See SP 800-56C for details concerning the appropriate form of *OtherInput*.

5.8.2 FixedInfo

The bit string *FixedInfo* **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 *FixedInfo*:

- 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.
- Agreed-upon encodings (as bit strings) of the values of one or more of the other parameters used as additional input to the KDM (e.g., *L*, *salt*, and/or *IV*).
- 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., the agreed-upon hash/MAC algorithms, the bit lengths of any MAC tags used for key confirmation, 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 *FixedInfo*, see Appendix B.

When *FixedInfo* is used, the meaning of each information item and each item's position within the *FixedInfo* bit string **shall** be specified. In addition, each item of information included in *FixedInfo* **shall** be unambiguously represented. For example, each item of information could take the form of a fixed-length bit string, or, if greater flexibility is needed, an item of information could be represented in a *Datalen* || *Data* format, 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 for *FixedInfo*, as specified in Section 5.8.2.1.2.

SP 800-56C specifies both one-step key-derivation methods (i.e., key-derivation functions) and two-step key-derivation methods (i.e., key-derivation procedures). The following subsections discuss possibilities for the form and format of *FixedInfo* when it is used by those **approved** key-derivation methods.

5.8.2.1 One-step Key Derivation

Recommended formats for *FixedInfo* when used by a one-step key-derivation method are specified in Sections 5.8.2.1.1 and 5.8.2.1.2. One of those two formats **should** be used by a one-step key-derivation method specified in SP 800-56C when the auxiliary function employed is H = *hash*.

When *FixedInfo* is included during the key-derivation process, and 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 non-null 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 non-null 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 non-null 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 **shall** be included in this field (see Sections 6.2.1.5 and 6.3.3).

SuppPubInfo: An optional subfield that contains additional, mutually known public information (e.g., *L*, 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). While an implementation may be capable of including this subfield, the subfield may be null for a given transaction.

SuppPrivInfo: An optional subfield that contains additional, mutually known private information (e.g., a shared secret symmetric key that has been communicated through a separate channel or established by other means). While an implementation may be capable of including this subfield, the subfield may be *Null* for a given transaction.

5.8.2.1.1 The Concatenation Format for *FixedInfo*

This section specifies the concatenation format for *FixedInfo*. 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 one-step key-derivation method specified in SP 800-56C is used with H = *hash* as the auxiliary function and this concatenation format for *FixedInfo*, the resulting key-derivation method is the Concatenation Key-Derivation Function specified in the original version of SP 800-56A.

For this format, *FixedInfo* 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 5.8.2.

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 *FixedInfo* **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 key-derivation method) **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 value that indicates the length (in bytes) of *Data*; the use of this form for the information allows parties U and V to omit an 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 *SuppPrivInfo* and/or *SuppPubInfo* subfield(s), 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* values.

5.8.2.1.2 The ASN.1 Format for *FixedInfo*

The ASN.1 format for *FixedInfo* 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 one-step key-derivation method specified in SP 800-56C is used with H = *hash* as the auxiliary function and this ASN.1 format for *FixedInfo*, 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, *FixedInfo* 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 5.8.2. A protocol using this format for *FixedInfo* **shall** specify the type, ordering and number of distinct items of information included in each of the (three, four, or five) subfields employed.

5.8.2.2 Two-step Key-Derivation (Extraction-then-Expansion)

For the two-step key-derivation method specified in SP 800-56C, *FixedInfo* is a bit string that contains component data fields such as a *Label*, *Context* information, and $[L]_2$, where:

- *Label* is a binary string that identifies the purpose of the derived keying material. The encoding method for the label is defined in a larger context, for example, in a protocol using the derivation method.
- *Context* is a binary string containing information relating to the derived keying material. Section 5.8.2 provides a list of context-specific information that may be appropriate for the inclusion in this string.
- $[L]_2$ is a binary string that specifies the length (in bits) of the keying material to be derived.

Different orderings of the component data fields of *FixedInfo* may be used, and one or more of the data fields may be combined (or omitted under certain circumstances). See Section 5 in SP 800-56C, and Sections 5, 7.4, 7.5 and 7.6 in SP 800-108 for details

5.8.2.3 Other Formats for *FixedInfo*

Formats other than those provided in Sections 5.8.2.1 and 5.8.2.2 (e.g., those providing the items of information in a different arrangement) may be used for *FixedInfo*, but context-specific information **should** be included (see the discussion in Section 5.8.2). This Recommendation makes no statement as to the adequacy of other formats.

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 Sections 5.9.1 and 5.9.2 for keying material derived during the execution of a key-agreement scheme, and in Section 7.2 for keying material transported during a key-transport scheme.

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 methods). *MacKey* and certain context-specific *MacData* (see step 2 below) 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 a *MacTag* 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 **shall** have an identifier, chosen in accordance with the assumptions stated for the 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; that nonce will also be used as part of the ephemeral data input to the MAC tag computations performed during key conformation. 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. The ephemeral data is 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 6.2.1.5.2, party V only contributes a static key pair; in this case, $EphemData_V$ can be *Null*.

     - When $EphemData_i$, (where *i* is *P* or *R*) is an ephemeral public key, the public key $EphemPubKey_i$ is a byte string determined as follows:

       For FFC schemes, *i*'s ephemeral public key, t_i, is converted from a field element in GF*(p)* to a byte string by representing the field element as an integer in the interval $[2,\ p - 2]$, and then converting the integer to a byte string as specified in Appendix C.1.

For ECC schemes, the coordinates of *i*'s ephemeral public key, $Q_{e,i}$, are converted from field elements to byte strings as specified in Appendix C.2 and concatenated (with the *x* coordinate first) to form a single byte string.

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

The content of each of the components that are concatenated to form $MacData_P$ **shall** be precisely defined and unambiguously represented. A component's content may be 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 could also be satisfied by using a specific ASN.1 DER encoding of each component. It is imperative that the provider and recipient have agreed upon the content and format that will be used for each component of $MacData_P$.

3. After computing the shared secret *Z* and applying the key-derivation method to obtain *DerivedKeyingMaterial* (see Section 5.8 and SP 800-56C), the provider uses agreed-upon bit lengths to parse *DerivedKeyingMaterial* into two parts, *MacKey* and *KeyData*, of the pre-agreed lengths:

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

4. Using an agreed-upon bit length *MacTagLen*, the provider computes $MacTag_P$ (see Sections 5.2.1 and 5.9.3):

   $$MacTag_P = T_{MacTagLen}[\text{MAC}\ (MacKey, MacData_P)],$$

   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 verifies that the computed $MacTag_P$ is equal to the value received from the provider. If the values are equal, 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 $MacData_P$. 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 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 party U providing $MacTag_U$ to recipient party V, and party V providing $MacTag_V$ to recipient party U) during the same key-agreement transaction. 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; the nonce will also be used as part of the ephemeral data input to the MAC tag computations performed during key conformation. 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.

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 Selecting the MAC and Other Key-Confirmation Parameters

Key confirmation as specified in this Recommendation requires that a *MacKey* of an appropriate length be generated as part of the derived keying material (see Section 5.9.1). The *MacKey* is then used with a MAC algorithm to generate a MAC; the length of the MAC output by the MAC algorithm is *MacOutputLen* bits. The MAC is subsequently used to form a MAC tag (see Section 5.9.1 for the generation of the MAC and Section 5.2.1 for the formation of the MAC tag from the MAC).

Table 5 provides a list of **approved** MAC algorithms for key confirmation and the security strengths that each can support, along with the corresponding value of *MacOutputLen* and permissible *MacKey* lengths for each MAC algorithm.

Table 5: Approved MAC Algorithms for Key Confirmation.

MAC Algorithm	*MacOutputLen* (in bits)	Permissable *MacKey* Lengths (μ bits)	Supported Security Strengths for Key Conformation
HMAC(SHA-1)	160	$112 \le \mu \le 512$ ($\mu \ge s$ is recommended)	112, 128, 192, 256
HMAC(SHA-224)	224		
HMAC(SHA-256)	256		
HMAC(SHA-512/224)	224		
HMAC(SHA-512/256)	256		
HMAC(SHA-384)	384		
HMAC(SHA-512)	512		
HMAC(SHA3-224)	224		
HMAC(SHA3-256)	256		
HMAC(SHA3-384)	384		
HMAC(SHA3-512)	512		
KMAC128	Choose *MacOutputLen* L, $L \le 2^{2040} – 1$ (see * below)		112, 128
KMAC256			112, 128, 192, 384, 256
AES-128-CMAC	128	$\mu = 128$	112, 128
AES-192-CMAC	128	$\mu = 192$	112, 128, 192
AES-256-CMAC	128	$\mu = 256$	112, 128, 192, 256

* Although KMAC128 and KMAC256 can accommodate *MacOutputLen* values as large as $2^{2040} – 1$, practical considerations dictate that the lengths of transmitted MAC tags be limited to sizes that are more realistic and commensurate with the actual performance/security requirements of the relying applications.

Note that Table 5 requires a minimum *MacKey* length of 112 bits, but recommends that a *MacKey* length of at least *s* bits be used, where *s* is the targeted security strength of the preceding steps of the key-establishment scheme. The lower bound for the *MacKey* length is set to 112 bits even when the targeted security strength for the key-establishment transaction is greater than 112 bits because, for key confirmation, each *MacKey* is used only once, and offline attacks are not considered to be a threat. Note that upper bounds have been placed on the *MacKey* lengths that are stricter than those appearing in the MAC algorithm specifications. In the case of HMAC, if *MacKey* is longer than the input block length, it would be hashed down to *MacOutputLen* bits during the HMAC computation (see step 2 in Table 1 of FIPS 198); making *MacKey* longer than the input block length would not be an efficient way of using the derived keying material, from which *MacKey* is obtained.

For the same reason, any **approved** MAC algorithm is allowed for key confirmation for the range of acceptable security strengths. However, the MAC algorithm **shall** be selected from among those capable of supporting a security strength that is at least as strong as the targeted key-establishment security strength *s*.

The length of the MAC tag also needs to be selected for key confirmation. Note that in many cases, the length of the MAC tag (*MacTagLen*) has been selected by the protocol in which the key-establishment is conducted. This Recommendation requires that *MacTagLen* be at least 64 bits, and its maximum length be no more than the *MacOutputLen* for the MAC algorithm selected for key confirmation. The 64-bit minimum for the MAC tag length assumes that the protocol imposes a limit on the number of retries for key confirmation.

6. Key Agreement

This Recommendation provides three categories of key-agreement schemes (see Table 6). 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 6: 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 7). In subcategory C(*i*e, *j*s), 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 7: 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.

Category	Subcategory
	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 8). The names of these schemes are taken from ANS X9.42 and ANS X9.63.

Table 8: 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)

Category	Subcategory	Primitive	Scheme	Notation
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 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 static keys used 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) scheme. 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.

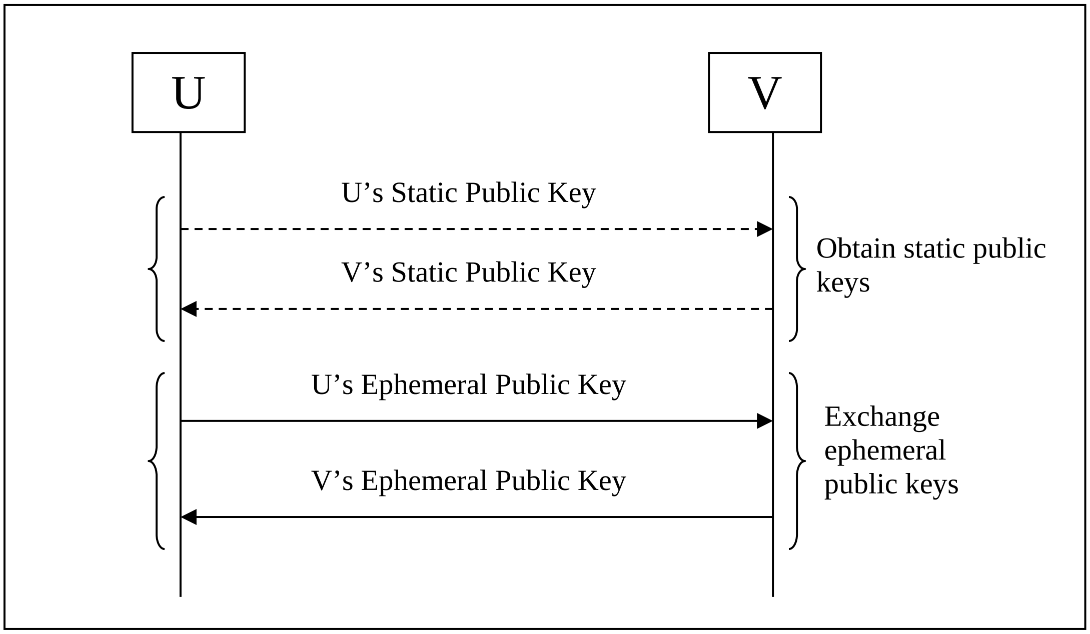

Figure 4: C(2e, 2s) schemes: each party contributes a static and an ephemeral key pair

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 are **approved** for use (see Section 5.5.1). For FFC schemes, $D = (p, q, g\{, SEED, counter\})$; 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, 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, including the lengths of *MacKey* and *MacTag*, as specified 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.

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 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, the dhHybrid1 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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party V. Receive an ephemeral public key t_V (purportedly) from party V. If t_V is not received, destroy the ephemeral private key r_U, then output an error indicator, and exit this process 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; then, output an error indicator, and exit this process 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, party U's static private key x_U, and party 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, and destroy the results of all intermediate calculations used in the attempted computation of Z_s; then output an error indicator, and exit this process 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, party U's ephemeral private key r_U, and party 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,, and destroy the results of all intermediate calculations used in the attempted computation of Z_e; then, output an error indicator, and exit this process 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 with the specified length from the shared secret value Z and other input (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, then output an error indicator, and exit this process without performing the remaining actions.

7. 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.

8. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material 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 9.

Table 9: dhHybrid1 key-agreement scheme summary

	Party U	**Party V**
Domain parameters	$D = (p, q, g\{, SEED, counter\})$	$D = (p, q, g\{, SEED, counter\})$

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 $Z = Z_e \parallel Z_s$	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 $Z = Z_e \parallel Z_s$
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 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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party V. Receive an ephemeral public key $Q_{e,V}$ (purportedly) from party V. If $Q_{e,V}$ is not received, destroy the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process 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 $d_{e,U}$, then output an error indicator, and exit this process 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, party U's static private key $d_{s,U}$, and party 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 $d_{e,U}$, and destroy the results of all intermediate calculations used in the attempted computation of Z_s; then output an error indicator, and exit this process 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, party U's ephemeral private key $d_{e,U}$, and party 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 $d_{e,U}$, and destroy the results of all intermediate calculations used in the attempted computation of Z_e; then output an error indicator, and exit this process 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 with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z and the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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 ($d_{e,U}$, $Q_{e,U}$) 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 $d_{e,U}$ **shall** be destroyed (see step 7 above).

The Full Unified Model is summarized in Table 10.

Table 10: 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 private key $d_{e,V}$

	Ephemeral public key $Q_{e,U}$	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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party V. Receive an ephemeral public key t_V (purportedly) from party V. If t_V is not received, destroy the ephemeral private key r_U; then output an error indicator, and exit this process 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; then output an error indicator, and exit this process 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, party U's static private key x_U, party V's static public key y_V, party U's ephemeral private key r_U, party U's ephemeral public key t_U, and party 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, and destroy

the results of all intermediate calculations used in the attempted computation of Z; then output an error indicator, and exit this process without performing the remaining actions.

4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z and other input (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; then output an error indicator, and exit this process without performing the remaining actions.
5. 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.
6. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material 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 11.

Table 11: MQV2 key-agreement scheme summary

	Party U	**Party V**
Domain parameters	$D = (p, q, g\{, SEED, counter\})$	$D = (p, q, g\{, SEED, counter\})$
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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party V. Receive an ephemeral public key $Q_{e,V}$ (purportedly) from party V. If $Q_{e,V}$ is not received, destroy the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process 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 $d_{e,U}$; then output an error indicator, and exit this process 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, party U's static private key $d_{s,U}$, party V's static public key $Q_{s,V}$, party U's ephemeral private key $d_{e,U}$, party U's ephemeral public key $Q_{e,U}$, and party 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 $d_{e,U}$, and destroy the results of all intermediate calculations used in the attempted computation of Z; then output an error indicator, and exit this process without performing the remaining actions.

4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z and the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process without performing the remaining actions.

5. 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 $d_{e,U}$.

6. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material 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 ($d_{e,U}$, $Q_{e,U}$) 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 $d_{e,U}$ **shall** be destroyed (see step 5 above).

The Full MQV is summarized in Table 12.

Table 12: 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 Party U to Party 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.

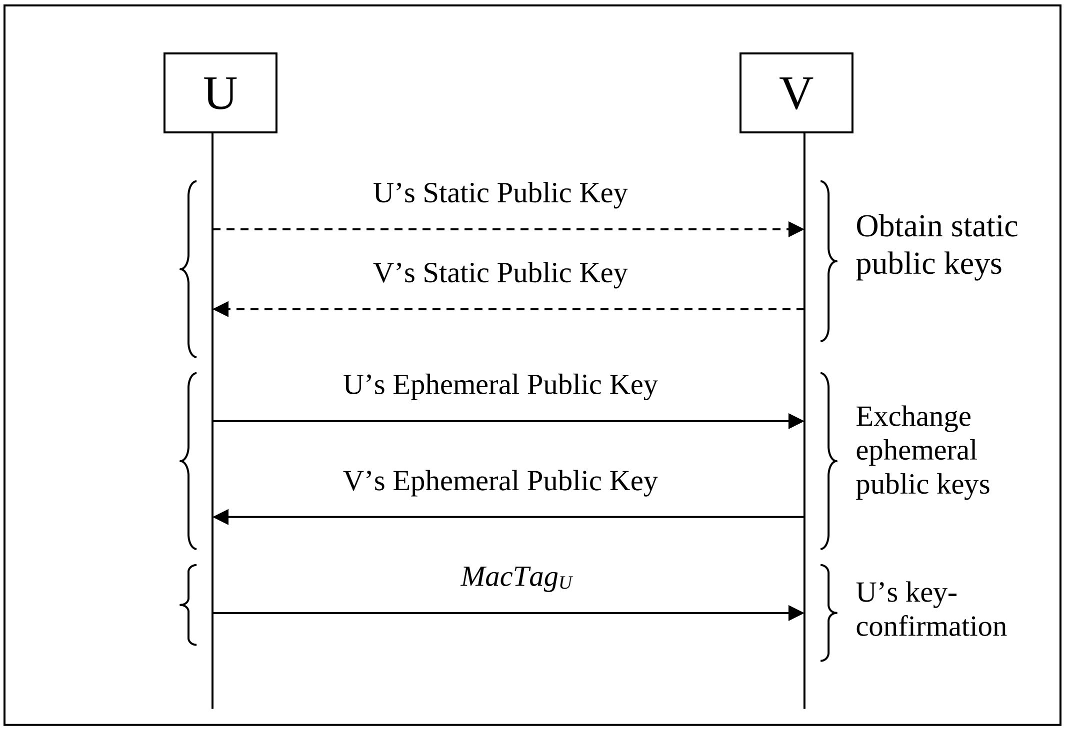

Figure 5: C(2e, 2s) scheme with unilateral key confirmation from party U to party V

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

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

Party U provides $MacTag_U$ to party 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 party U.

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

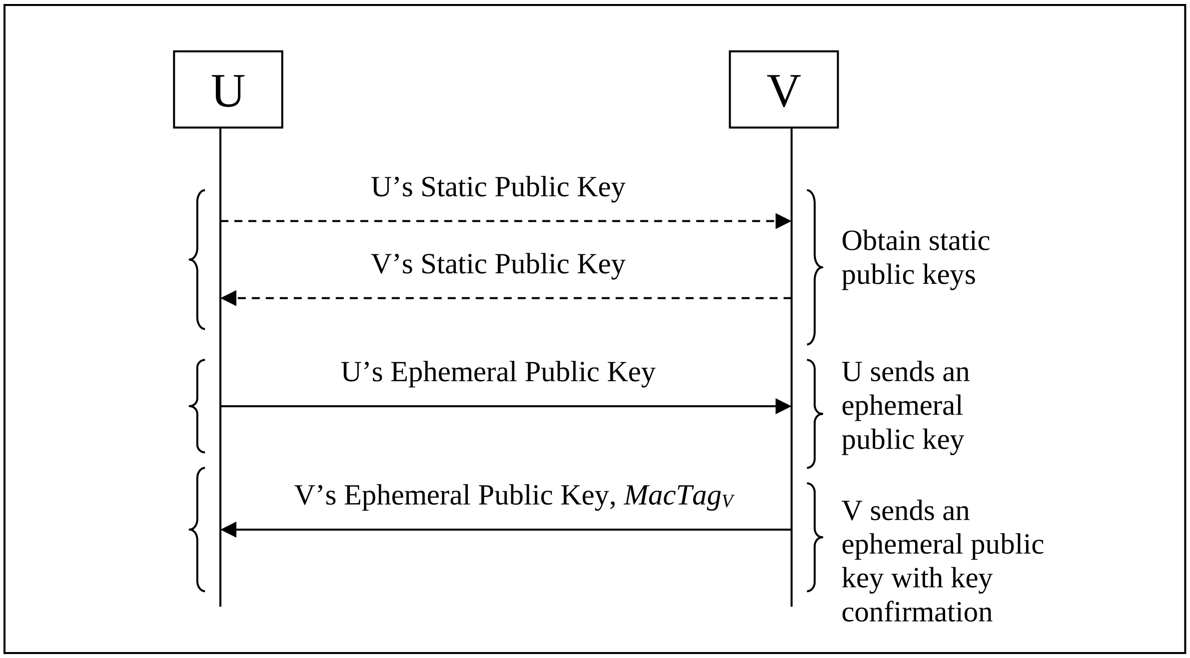

Figure 6: C(2e, 2s) scheme with unilateral key confirmation from party V to party U

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

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

Party V provides $MacTag_V$ to party 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

$$MacData_V = \text{“KC\_1\_V”} \,\|\, ID_V \,\|\, ID_U \,\|\, EphemPubKey_V \,\|\, EphemPubKey_U \,\{\|\, Text_V\}.$$

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 party 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) assurance that the other party has actively participated in the process.

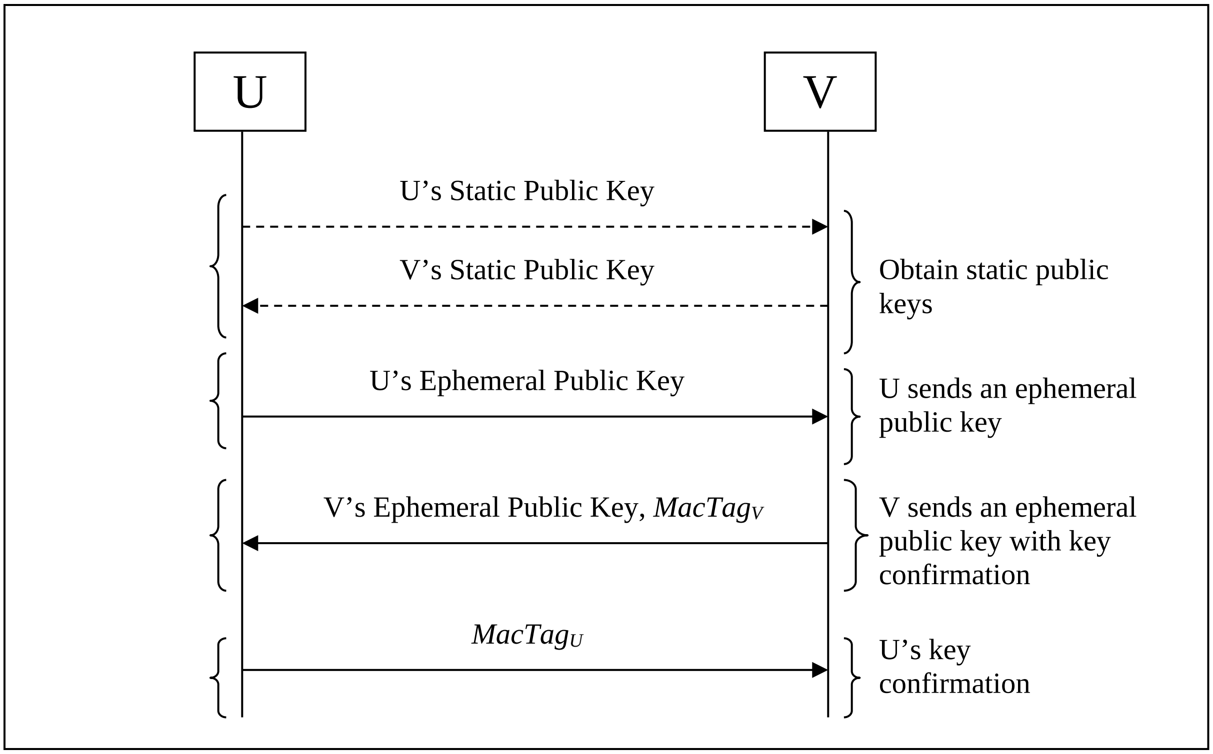

Figure 7: C(2e, 2s) scheme with bilateral key confirmation

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

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

Party V provides $MacTag_V$ to party 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 party V (and verified by party U) using

$$MacData_V = \text{“KC\_2\_V”} \parallel ID_V \parallel ID_U \parallel EphemPubKey_V \parallel EphemPubKey_U \{\parallel Text_V\}.$$

Party U provides $MacTag_U$ to party 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 party U (and verified by party V) using

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

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).

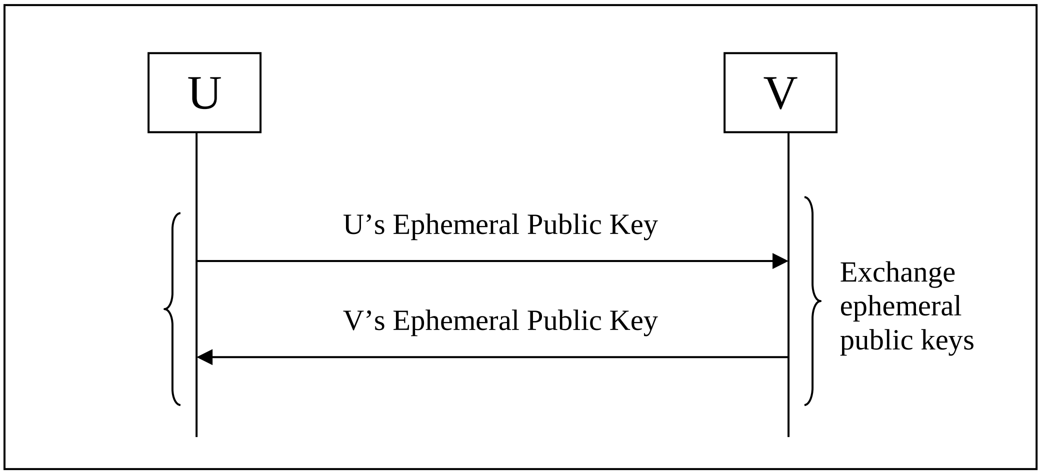

Figure 8: C(2e, 0s) schemes: each party contributes only an ephemeral key pair

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 are either **approved** for use in the intended application (see Section 5.5.1). For FFC schemes, *D* = (*p*, *q*, *g*{, *SEED*, *counter*}); 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, the dhEphem 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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party V. Receive an ephemeral public key t_V (purportedly) from party V. If t_V is not received, destroy the ephemeral private key r_U; then output an error indicator, and exit this process 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; then output an error indicator, and exit this process 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, party U's ephemeral private key r_U, and party 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; then output an error indicator, and exit this process without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z; then output an error indicator, and exit this process without performing the remaining action.
5. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material or an error indicator.

dhEphem is summarized in Table 13.

Table 13: dhEphem key-agreement scheme summary

	Party U	**Party V**
Domain parameters	(p, q, g{, $SEED$, $counter$})	(p, q, g{, $SEED$, $counter$})
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
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) Scheme

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, the Ephemeral Unified Model 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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party V. Receive an ephemeral public key $Q_{e,V}$ (purportedly) from party V. If $Q_{e,V}$ is not received, destroy the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process 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 $d_{e,U}$; then output an error indicator, and exit this process 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*, party U's ephemeral private key $d_{e,U}$, and party 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*, then output an error indicator, and exit this process without performing the remaining actions.

4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z* and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of *Z*; then output an error indicator, and exit this process without performing the remaining action.

5. Destroy all copies of the shared secret *Z* and output the derived keying material.

Output: The derived keying material or an error indicator.

The Ephemeral Unified Model is summarized in Table 14.

Table 14: 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) Schemes

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) Schemes

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

Figure 9 depicts a typical flow for a C(1e, 2s) scheme. 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.

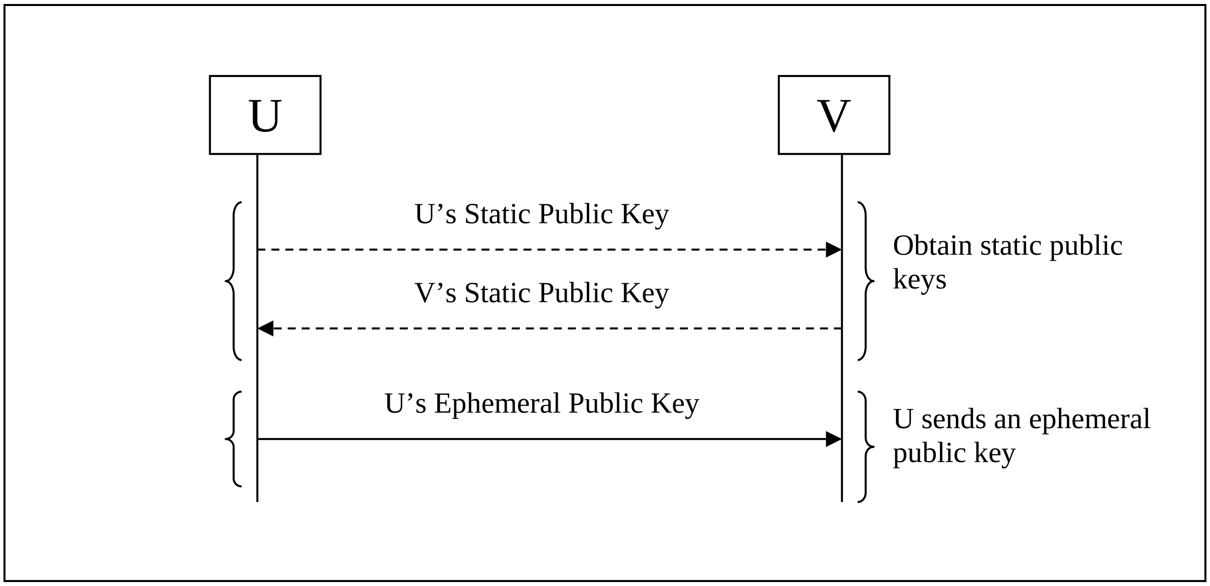

Figure 9: C(1e, 2s) schemes: party U contributes a static and an ephemeral key pair while party V contributes only a static key pair

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 are either **approved** for use in the intended application (see Section 5.5.1). For FFC schemes, *D* = (*p*, *q*, *g*{, *SEED*, *counter*}); 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. Each party has also 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, 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, including the lengths of *MacKey* and *MacTag* (see Section 5.9.3). If party V provides key confirmation to party U, the parties have agreed upon the form of $Nonce_V$, which **should** be a random nonce (see Section 5.4).

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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party 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, party U's static private key x_U, and party 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, and destroy the results of all intermediate calculations used in the attempted computation of Z_s; then output an error indicator, and exit this process 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, party U's ephemeral private key r_U, and party 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, and destroy the results of all intermediate calculations used in the attempted computation of Z_e; then output an error indicator, and exit this process 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 with the specified length from the shared secret value Z and other input (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; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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: Party V derives secret keying material as follows:

1. Receive an ephemeral public key t_U (purportedly) from party U. If t_U is not received, then output an error indicator, and exit this process 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, then output an error indicator, and exit this process 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, party V's static private key x_V, and party 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; then output an error indicator, and exit this process 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, party V's static private key x_V, and party U's ephemeral public key t_U. If this call to the FFC DH primitive outputs an error indicator, destroy Z_s, and destroy the results of all intermediate calculations used in the attempted computation of Z_e; then output an error indicator, and exit this process 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 with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z; then output an error indicator, and exit this process without performing the remaining action.
7. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material 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 15.

Table 15: dhHybridOneFlow key-agreement scheme summary

	Party U	**Party V**
Domain parameters	$(p, q, g\{, SEED, counter\})$	$(p, q, g\{, SEED, counter\})$
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 $Z = Z_e \parallel Z_s$	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 $Z = Z_e \parallel Z_s$
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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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, party U's static private key $d_{s,U}$, and party 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 $d_{e,U}$, and destroy the results of all intermediate calculations used in the attempted computation of Z_s; then output an error indicator, and exit this process without performing the remaining actions.

3. Use the ECC CDH primitive to derive a shared secret Z_e, from the set of domain parameters D, party U's ephemeral private key $d_{e,U}$, and party 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 $d_{e,U}$, and destroy the results of all intermediate calculations used in the attempted computation of Z_e; then output an error indicator, and exit this process 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 with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z and the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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 ($d_{e,U}$, $Q_{e,U}$) 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 $d_{e,U}$ **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: Party V derives secret keying material as follows:

1. Receive an ephemeral public key $Q_{e,U}$ (purportedly) from party U. If $Q_{e,U}$ is not received, then output an error indicator, and exit this process 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, then output an error indicator, and exit this process 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, party V's static private key $d_{s,V}$, and party 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; then output an error indicator, and exit this process 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, party V's static private key $d_{s,V}$, and party U's ephemeral public key $Q_{e,U}$. If this call to the ECC CDH primitive outputs an error indicator, destroy Z_s, and destroy the results of all intermediate calculations used in the attempted computation of Z_e; then output an error indicator, and exit this process without performing the remaining actions.
5. Compute the shared secret $Z = Z_e \mid\mid Z_s$. Destroy Z_e and Z_s.
6. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z; then output an error indicator, and exit this process without performing the remaining action.
7. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material 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 16.

Table 16: 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 $Z = Z_e \mid\mid Z_s$	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 $Z = Z_e \mid\mid Z_s$

	Party U	Party V
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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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*, party U's static private key x_U, party V's static public key y_V, party U's ephemeral private key r_U, party U's ephemeral public key t_U, and (for a second time) party 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, and destroy the results of all intermediate calculations used in the attempted computation of *Z*; then output an error indicator, and exit this process without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z* and other input (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; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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: Party V derives secret keying material as follows:

1. Receive an ephemeral public key t_U (purportedly) from party U. If t_U is not received, then output an error indicator, and exit this process 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, then output an error indicator, and exit 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, party V's static private key x_V, party U's static public key y_U, party V's static private key x_V (for a second time), party V's static public key y_V, and party 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; then output an error indicator, and exit this process without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z; then output an error indicator, and exit this process without performing the remaining action.
5. Destroy all copies of the shared secret Z and output *DerivedKeyingMaterial*.

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

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

MQV1 is summarized in Table 17.

Table 17: MQV1 Key-agreement scheme summary.

	Party U	**Party V**
Domain parameters	$(p, q, g\{, SEED, counter\})$	$(p, q, g\{, SEED, counter\})$
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	N/A

	Ephemeral public key t_U	
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 party U and party V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party 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*, party U's static private key $d_{s,U}$, party V's static public key $Q_{s,V}$, party U's ephemeral private key $d_{e,U}$, party U's ephemeral public key $Q_{e,U}$, and (for a second time) party 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 $d_{e,U}$, and destroy the results of all intermediate calculations used in the attempted computation of *Z*; then output an error indicator, and exit this process without performing the remaining actions.

3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z* and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of *Z* and the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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 ($d_{e,U}$, $Q_{e,U}$) 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 $d_{e,U}$ **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 shared secret keying material from *Z*.

Actions: Party V derives secret keying material as follows:

1. Receive an ephemeral public key $Q_{e,U}$ (purportedly) from party U. If $Q_{e,U}$ is not received, then output an error indicator, and exit this process 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, then output an error indicator, and exit 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*, party V's static private key $d_{s,V}$, party U's static public key $Q_{s,U}$, party V's static private key $d_{s,V}$ (for a second time), party V's static public key $Q_{s,V}$, and party 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*; then output an error indicator, and exit this process without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z* and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of *Z*; then output an error indicator, and exit this process without performing the remaining action.
5. Destroy all copies of the shared secret *Z* and output the derived keying material.

Output: The derived keying material or an error indicator.

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

The One-Pass MQV scheme is summarized in Table 18.

Table 18: 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 party V cannot act as a key-confirmation recipient unless a nonce ($Nonce_V$) is provided by party V to party U and is used (in addition to the shared secret Z) as input to the key-derivation method employed by the scheme. This would be accomplished by including (a copy of) $Nonce_V$ in the *OtherInput* provided to the KDM, as part of the *FixedInfo* (see Section 5.8), in addition to using (a copy of) $Nonce_V$ as the $EphemData_V$ employed in the *MacTag* computations for key confirmation.

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 Party U to Party 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 party V does not contribute an ephemeral public key during the key-agreement process, a nonce ($Nonce_V$) **shall** be provided by party V to party U and used (in addition to the shared secret Z) as input to the key-derivation method employed by the scheme. $Nonce_V$ is also 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.

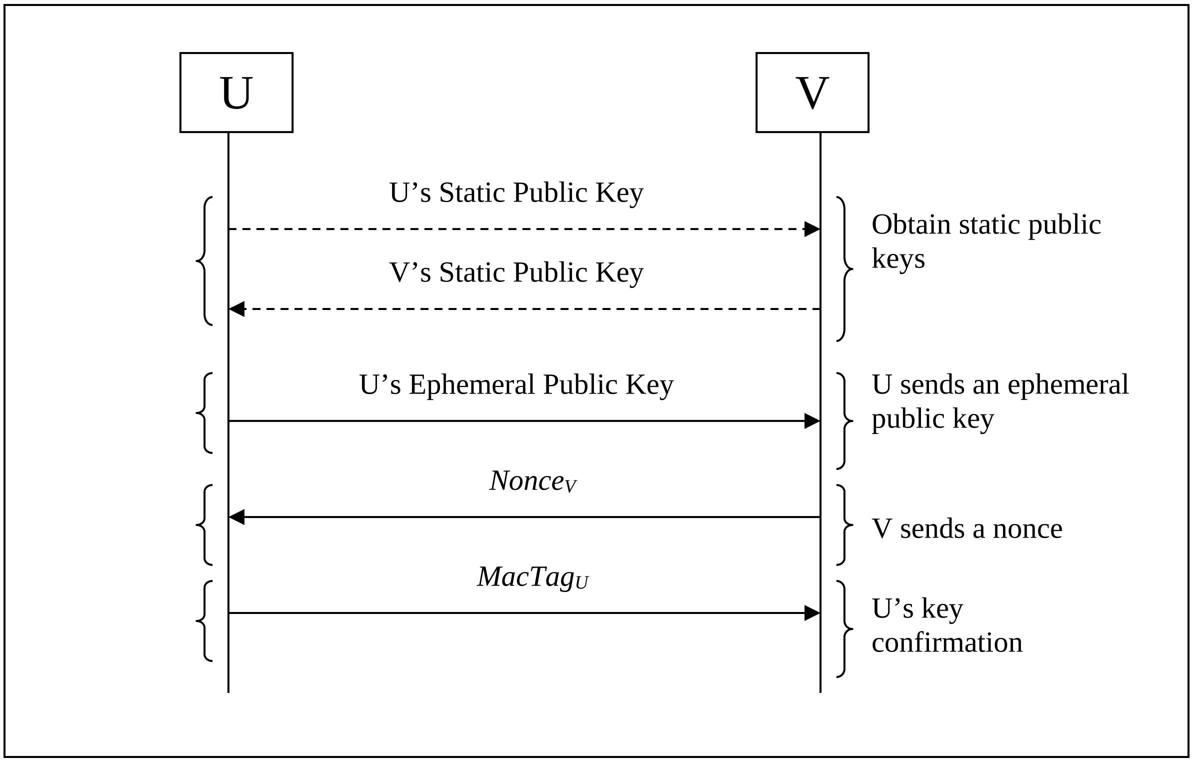

Figure 10: C(1e, 2s) scheme with unilateral key confirmation from party U to party V

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

$$EphemData_U = EphemPubKey_U, \text{ and } EphemData_V = Nonce_V.$$

Party U provides $MacTag_U$ to party 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

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

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 party U.

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

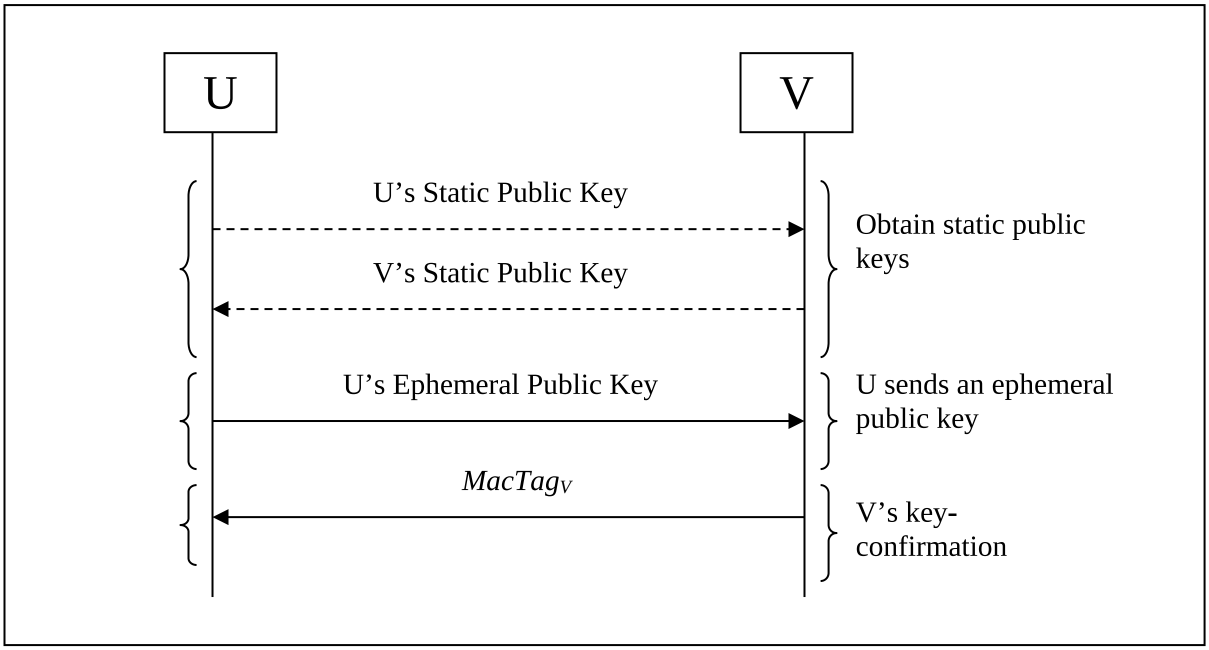

Figure 11: C(1e, 2s) scheme with unilateral key confirmation from party V to party U

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 party 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 party 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 to obtain bilateral key confirmation. Since party V does not contribute an ephemeral public key during the key-agreement process, a nonce ($Nonce_V$) **shall** be provided by party V to party U and used (in addition to the shared secret *Z*) as input to the key-derivation method employed by the scheme. $Nonce_V$ is also 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.

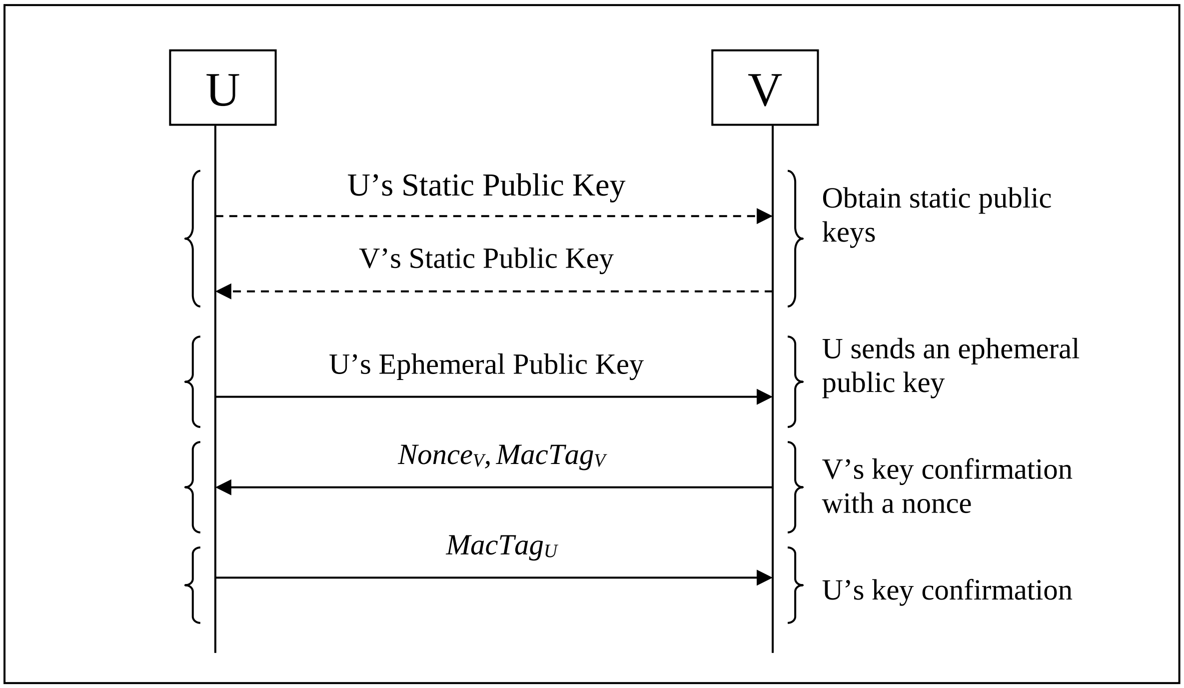

Figure 12: C(1e, 2s) scheme with bilateral key confirmation

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

$$EphemData_U = EphemPubKey_U \text{ and } EphemData_V = Nonce_V.$$

Party V provides $MacTag_V$ to party 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 party V (and verified by U) using

$$MacData_V = \text{“KC\_2\_V”} \parallel ID_V \parallel ID_U \parallel Nonce_V \parallel EphemPubKey_U \{\parallel Text_V\}.$$

Party U provides $MacTag_U$ to party 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 party U (and verified by party V) using

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

Note that in Figure 12 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 the 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).

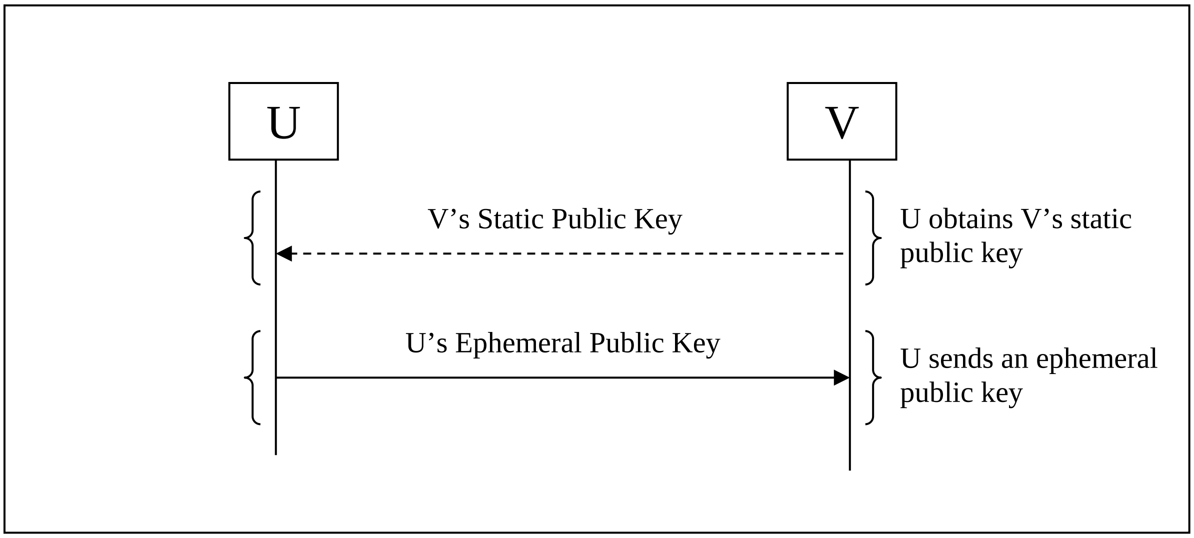

Figure 13: C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair

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 are either **approved** for use in the intended application (see Section 5.5.1). For FFC schemes, $D = (p, q, g\{, SEED, counter\})$; 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 also agreed upon an **approved** MAC and associated parameters, including the lengths of *MacKey* and *MacTag* (see 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 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 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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party 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*, party U's ephemeral private key r_U, and party 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, and destroy the results of all intermediate calculations used in the attempted computation of *Z*; then output an error indicator, and exit this process without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z* and other input (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; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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: Party V derives secret keying material as follows:

1. Receive an ephemeral public key t_U (purportedly) from party U. If t_U is not received, then output an error indicator, and exit this process 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, then output an error indicator, and exit this process 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*, party V's static private key x_V, and party 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*; then output an error indicator, and exit this process without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z* and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of *Z*; then output an error indicator, and exit this process without performing the remaining action.
5. Destroy all copies of the shared secret *Z* and output the derived keying material.

Output: The derived keying material 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 19.

Table 19: dhOneFlow key-agreement scheme summary

	Party U	Party V
Domain parameters	$(p, q, g\{, SEED, counter\})$	$(p, q, g\{, SEED, counter\})$
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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party 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, party U's ephemeral private key $d_{e,U}$, and party 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 $d_{e,U}$, and destroy the results of all intermediate

calculations used in the attempted computation of Z; then output an error indicator, and exit this process without performing the remaining actions.

3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z and the ephemeral private key $d_{e,U}$; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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 ($d_{e,U}$, $Q_{e,U}$) 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 $d_{e,U}$ **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: Party V derives secret keying material as follows:

1. Receive an ephemeral public key $Q_{e,U}$ (purportedly) from party U. If $Q_{e,U}$ is not received, then output an error indicator, and exit this process 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, then output an error indicator, and exit 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, party V's static private key $d_{s,V}$, and party 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; then output an error indicator, and exit this process without performing the remaining actions.
4. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z; then output an error indicator, and exit this process without performing the remaining action.
6. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material 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 20.

Table 20: 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 party V to party 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 Party V to Party 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.

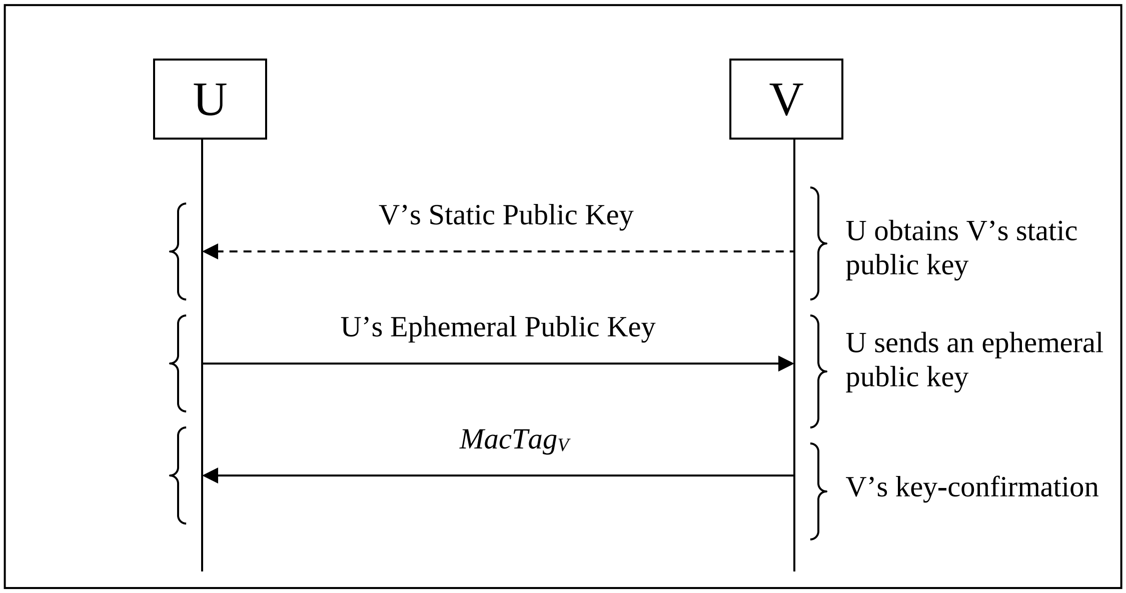

Figure 14: C(1e, 1s) scheme with unilateral key confirmation from party V to party U

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

$$EphemData_V = Null, \text{ and } EphemData_U = EphemPubKey_U.$$

Party V provides $MacTag_V$ to party 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

$$MacData_V = \text{“KC\_1\_V”} \parallel ID_V \parallel ID_U \parallel Null \parallel EphemPubKey_U \{\parallel Text_V\}.$$

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 key. A nonce, $Nonce_U$, is sent by party U to party V to ensure that the derived keying material is different for each key-establishment transaction. This would be accomplished by including (a copy of) $Nonce_U$ in the *OtherInput* provided to the KDM, as part of the *FixedInfo* (see Section 5.8). 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).

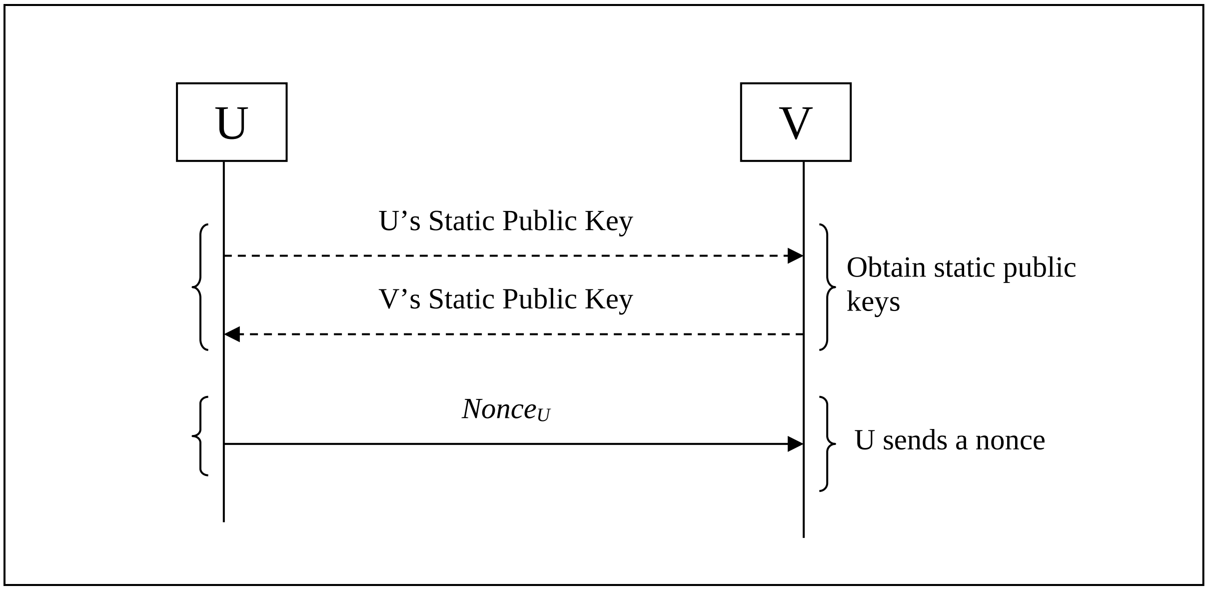

Figure 15: C(0e, 2s) schemes: each party contributes only a static key pair

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 are either **approved** for use in the intended application (see Section 5.5.1). For FFC schemes, *D* = (*p*, *q*, *g*{, *SEED*, *counter*}); 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 used with the method (e.g., a hash function) and other associated parameters to be used. In addition, the parties have agreed on the form of the nonce (see Section 5.4), which **should** be a random nonce.

4. If key confirmation is used, the parties have also agreed upon an **approved** MAC and associated parameters, including the lengths of *MacKey* and *MacTag* (see Section 5.9.3). If party V provides key confirmation to party U, the parties have agreed upon the form of $Nonce_V$, which **should** be a random nonce.

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.

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 5.6.3.2.

7. When an identifier is used to label a party during the key-agreement process, that identifier has a trusted association with 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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party 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*, party U's static private key x_U, and party V's static public key y_V. If the call to the FFC DH primitive outputs an error indicator, destroy $Nonce_U$, and destroy the results of all intermediate calculations used in the attempted computation of *Z*; then output an error indicator, and exit this process without performing the remaining actions.

3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z*, $Nonce_U$ and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of *Z*; then output an error indicator, and exit this process without performing the remaining actions.

4. Destroy all copies of the shared secret *Z* and output the derived keying material.

Output: The derived keying material 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).

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: Party V derives secret keying material as follows:

1. Obtain party U's nonce, $Nonce_U$, from party U. If $Nonce_U$ is not available, then output an error indicator, and exit this process 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, party V's static private key x_V, and party 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; then output an error indicator, and exit this process without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value Z, $Nonce_U$, and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of Z; then output an error indicator, and exit this process without performing the remaining action.
4. Destroy all copies of the shared secret Z and output the derived keying material.

Output: The derived keying material 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 21.

Table 21: dhStatic key-agreement scheme summary

	Party U	**Party V**
Domain parameters	$(p, q, g\{, SEED, counter\})$	$(p, q, g\{, SEED, counter\})$
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 Z and $Nonce_U$ 2. Destroy Z	1. Compute *DerivedKeyingMaterial* using Z and $Nonce_U$ 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 parties U and V must use identical orderings of the bit strings that are input to the key-derivation method.

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: Party 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 party 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*, party U's static private key $d_{s,U}$, and party 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*; then output an error indicator, and exit this process without performing the remaining actions.
3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z*, $Nonce_U$, and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of *Z*; then output an error indicator, and exit this process 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 the derived keying material.

Output: The derived keying material 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).

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: Party V derives secret keying material as follows:

1. Obtain party U's nonce, $Nonce_U$, from party U. If $Nonce_U$ is not available, then output an error indicator, and exit this process without performing the remaining actions.
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*, party V's static private key $d_{s,V}$, and party 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*; then output an error indicator, and exit this process without performing the remaining actions.

3. Use the agreed-upon key-derivation method to derive secret keying material with the specified length from the shared secret value *Z*, $Nonce_U$, and other input (see Section 5.8). If the key-derivation method outputs an error indicator, destroy all copies of *Z*; then output an error indicator, and exit this process without performing the remaining action.

4. Destroy all copies of the shared secret *Z* and output the derived keying material.

Output: The derived keying material 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 22.

Table 22: 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 unless a nonce ($Nonce_V$) is provided by party V to party U and is used (in addition to the shared secret *Z*) as input to the key-derivation method employed by the scheme. This would be accomplished by including (a copy of) $Nonce_V$ in the *OtherInput* provided to the KDM, as part of the *FixedInfo* (see Section 5.8), in addition to using (a copy of) $Nonce_V$ as the $EphemData_V$ employed in the *MacTag* computations for key confirmation.

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 Party U to Party 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 party V to party 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 also 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.

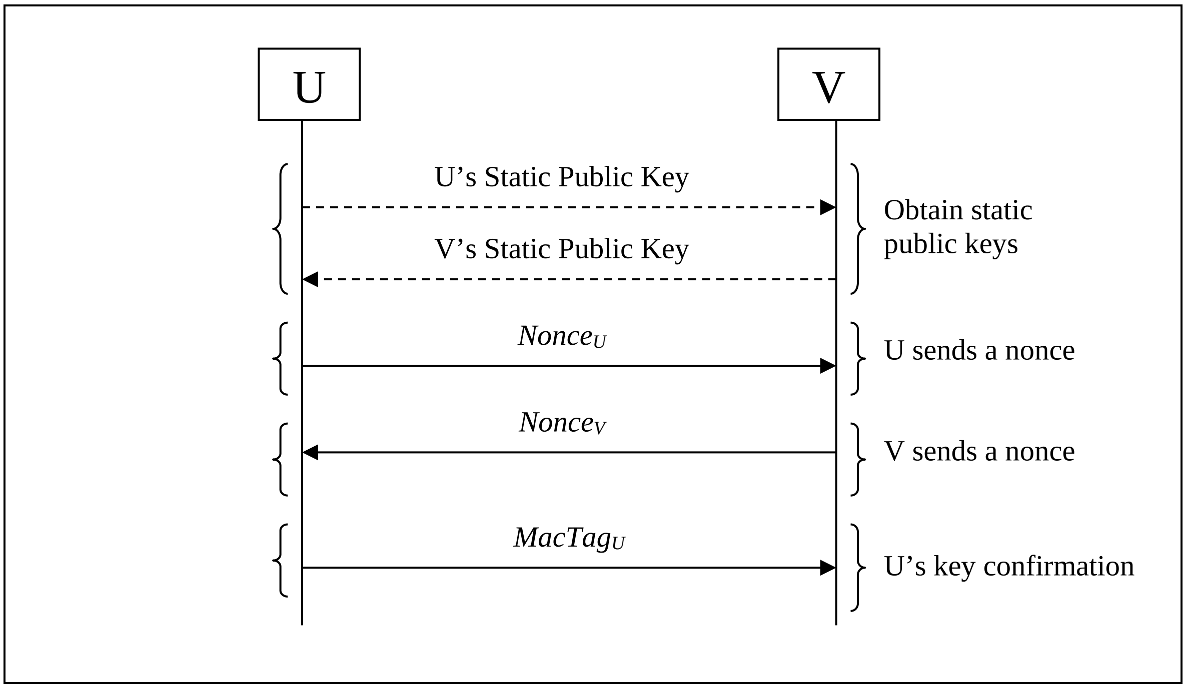

Figure 16: C(0e, 2s) scheme with unilateral key confirmation from party U to party V

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

$$EphemData_U = Nonce_U, \text{ and } EphemData_V = Nonce_V.$$

Party U provides $MacTag_U$ to party 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

$$MacData_U = \text{“KC\_1\_U”} \parallel ID_U \parallel ID_V \parallel Nonce_U \parallel Nonce_V \{\parallel Text_U\}.$$

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 party U.

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

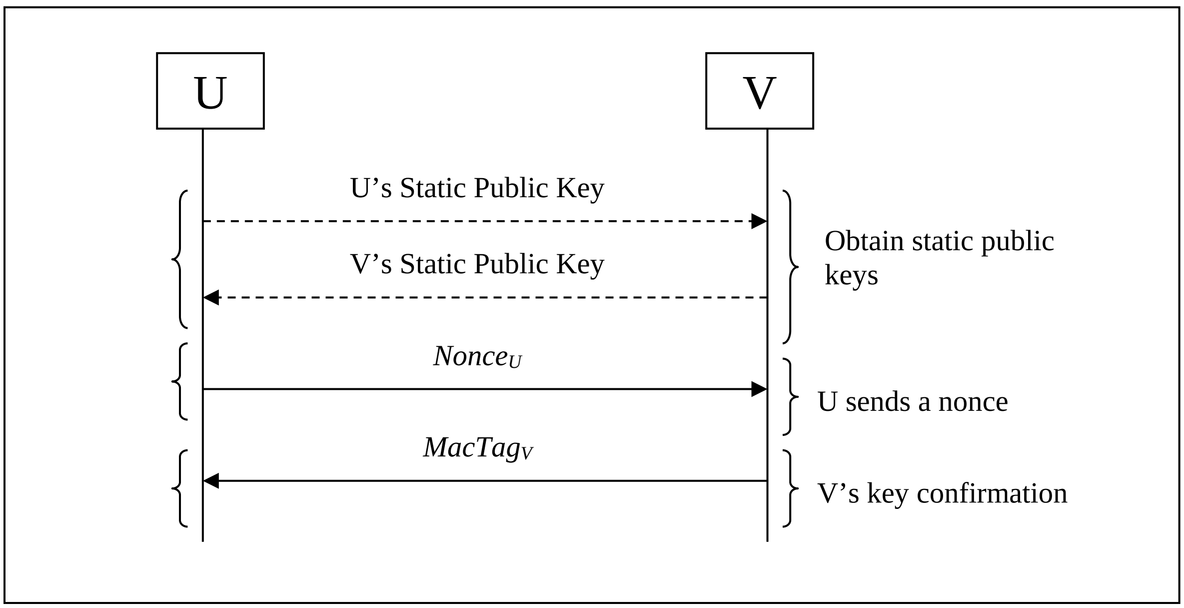

Figure 17: C(0e, 2s) scheme with unilateral key confirmation from party V to party U

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

$$EphemData_V = Null, \text{ and } EphemData_U = Nonce_U.$$

Party V provides $MacTag_V$ to party 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

$$MacData_V = \text{“KC\_1\_V”} \parallel ID_V \parallel ID_U \parallel Null \parallel Nonce_U \{\parallel Text_V\}.$$

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 party V.

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

Figure 18 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 party V to party 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 also 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.

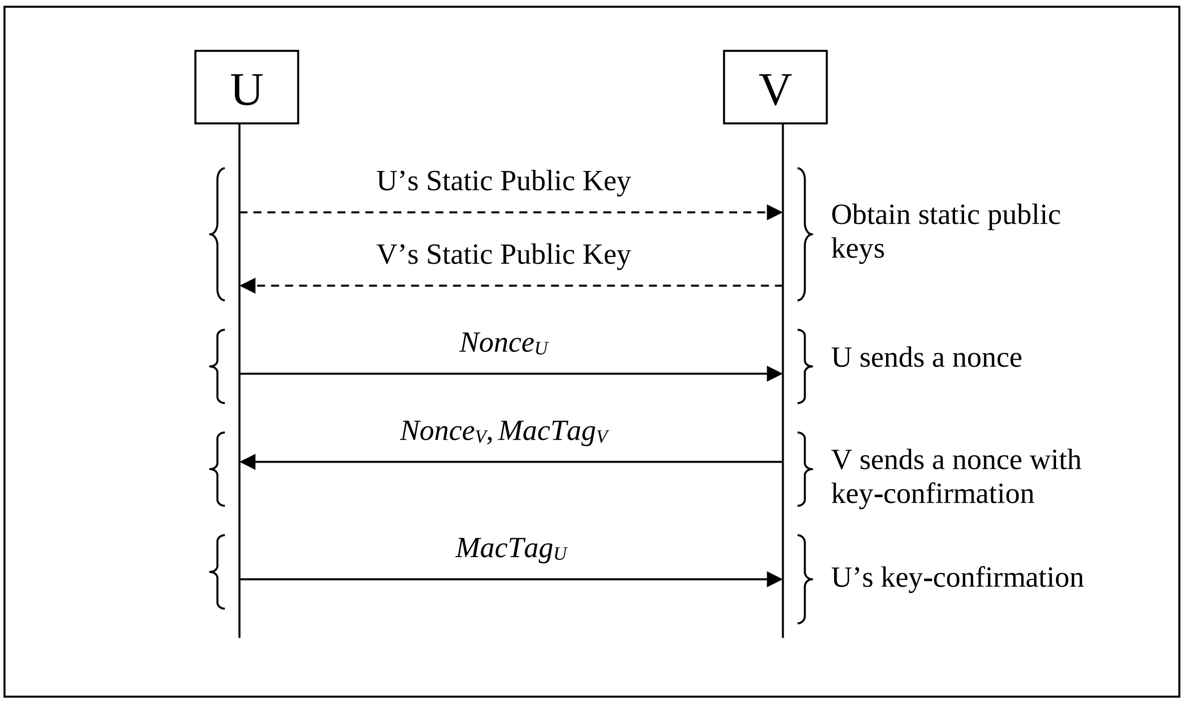

Figure 18: C(0e, 2s) scheme with bilateral key confirmation

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

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

Party V provides $MacTag_V$ to party 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 party V (and verified by party U) using

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

Party U provides $MacTag_U$ to party 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 party U (and verified by party V) using

$$MacData_U = \text{"KC\_2\_U"} \,\|\, ID_U \,\|\, ID_V \,\|\, Nonce_U \,\|\, Nonce_V \,\{\|\, Text_U\}.$$

Note that in Figure 18, 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 can also 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 (Alternative 1)

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** DLC-based key-agreement scheme. This key **shall** be used by party U to wrap secret keying material using an **approved** key-wrapping algorithm, based on the use of AES-128, AES-192 or AES-256. Three methods of key wrapping are **approved** for DLC-based key transport: CCM, KW and KWP; CCM is specified in SP 800-38C, while KW and KWP are specified in SP 800-38F.

The wrapped keying material is 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).

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 (e.g., the use of C(2e, 0s) schemes with or without signed ephemeral pubic keys), 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.

Assumptions: In order to execute a DLC key-transport scheme in compliance with this Recommendation, the following assumptions **shall** be true:

1. All assumptions for the key-agreement scheme used **shall** be true (see Sections 6.1.1, 6.2.1, 6.2.2 and 6.3).

2. The sender and receiver have agreed upon an **approved** AES variant (i.e., AES-128, AES-192 or AES-256) and key-wrapping method (i.e., either CCM, KW or KWP). The key-wrapping method **shall** protect the transported keying material at a security strength that is equal to or greater than the target security strength of the applicable key-establishment scheme.

   If the CCM mode is used during key wrapping, the sender and receiver have agreed on the counter-generation function, the formatting function, and *TLen*, the bit length of the CBC-MAC tag to be produced during the key-wrapping operation (see Sections 7.1.1 and 7.1.2).

---

[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.

If the KW or KWP mode is used for key wrapping, the sender and receiver have agreed on the valid plaintext lengths to be used during key wrapping (see Sections 7.1.3 and 7.1.4).

3. If the CCM mode will be used for key wrapping, prior to or during the key-establishment process, the parties have either agreed upon the format and content of the additional input *A* (a string to be cryptographically bound to the transported keying material so that the cipher is a cryptographic function of both values), or agreed that *A* will be the empty string. Note that for the KW and KWP modes, additional input is not accommodated.
4. If the CCM mode is used for key wrapping, either party U and party V **shall** have agreed on the MAC-tag length (*Tlen*) for the key-wrapping process, or party U **shall** send the CBC-MAC-tag length to party V, along with the wrapped keying material.
5. The sender and receiver have agreed on whether or not key confirmation will be used following the transport of the wrapped keying material. If key confirmation is used, the parties have also agreed upon an **approved** MAC algorithm and associated parameters, including the lengths of *MacKey* and *MacTag*, as specified in Section 5.9.3).
6. Prior to or during the key-establishment process, the keying material to be transported has been (or will be) determined.

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 key-wrapping key *KWK* that will subsequently be used by party U for key transport. 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 *KWK* from the *DerivedKeyingMaterial*.
3. Party U selects secret keying material, *KM*, to transport to party V, the receiver. If key confirmation is to be performed following key transport, this *KM* **shall** include a fresh (i.e., not previously used) *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 *WrappedKM* = KWA.WRAP(*KWK*, *KM*, *OtherKWAInput*) using an **approved** key-wrapping algorithm; see Sections 7.1.1 and 7.1.3.
5. Party U sends *WrappedKM* to party V, along with any other necessary information (e.g., *OtherKWAInput*).
6. Party V receives *WrappedKM* and *OtherKWAInput* from party U.
7. Party V obtains the *KWK* from the *DerivedKeyingMaterial*.

8. Party V calculates *KM* = KWA.UNWRAP(*KWK*, *WrappedKM*, *OtherKWAInput*) using the key-unwrapping algorithm that corresponds to the key wrapping algorithm used in step 4; see Sections 7.1.2 and 7.1.4.

9. If key confirmation is to be performed subsequent to key transport to provide assurance to party U that the correct plaintext keying material *KM* 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.

7.1.1 Key-Wrapping using AES-CCM

The input to the CCM mode specified in SP 800-38C includes a nonce, *Nonce*, additional input[11] *A* and the keying material to be wrapped[12], *KM*; the additional input could be a null string. See Appendix A.1 in SP 800-38C for restrictions on the (individual and combined) lengths of the nonce, the additional input and the keying material to be wrapped.

Also required for the CCM mode is *TLen*, the bit length of the MAC tag, *T*, to be produced; see Appendix B.2 in SP 800-38C for guidance on the selection of *TLen*. The wrapping operation uses a key-wrapping key[13] *KWK* to produce the ciphertext, *WrappedKM*, based on the input (i.e., a nonce, any additional input, *A*, and the keying material KM to be wrapped). Note that *WrappedKM* includes the MAC tag.

The chosen *Nonce*, the value of *TLen* and the additional input, *A*, **shall** be available to both party U and party V (e.g., by an exchange of information and/or using information already known by both parties). For recommendations concerning the types of information that may be appropriate for inclusion in the additional input *A*, see Section 5.8.2. That section

[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).

[11] Called associated data in SP 800-38C.

[12] Called the payload *P* in SP 800-38C.

[13] Called *K* in SP 800-38C.

discusses the content of *FixedInfo*, whose role in key derivation is analogous to the role played by *A* in this key-wrapping variant (namely, binding the established keying material to the context of the key-establishment transaction).

Party U, who wraps the keying material, **shall** provide the nonce to the receiving party, party V.

The key-wrapping operation using CCM is:

Function call: KWA.WRAP(*KWK*, *KM*, *OtherKWAInput*)
Input:

1. *KWK*: The key-wrapping key; a 128-, 192- or 256-bit key.
2. *KM*: The keying material to be wrapped; a bit string.
3. *OtherKWAInput*:
   a) *Nonce*: A nonce, as specified in Section 5.4; a bit string.
   b) *TLen*: The bit length of the MAC tag *T* to be generated; an integer.
   c) *A*: Additional input; a (possibly empty) byte string.

Process:

1. Check that the following conditions are satisfied:
   - The length of the *KWK* is the agreed-upon length (see assumption 2),
   - The value of *TLen* is valid for AES-CCM, and
   - The lengths of *KM*, *Nonce*, and *A* are valid for the CCM mode[14].

   If any of these conditions is not satisfied, then return an error indicator, and exit without further processing.
2. *WrappedKM* = **CCM.Encrypt**(*KWK*, *TLen*, *Nonce*, *KM*, *A*).
3. Return *WrappedKM*.

Output:
The ciphertext *WrappedKM* (a bit string) or an error indicator.

Note that the inputs to the **CCM.Encrypt** operation in process step 2 do not exactly match the specification of the Generation-Encryption process in SP 800-38C, in which (the equivalents of) *KWK* and *TLen* are listed as prerequisites, while the nonce, additional input and keying material to be wrapped are listed as inputs.

A routine that implements this operation **shall** destroy any local copies of sensitive input values (including *KWK*, *KM*, and any sensitive portions of *A*), as well as any other potentially sensitive locally stored values used or produced during its execution. (The **CCM.Encrypt** routine **should** do the same.) Their destruction **shall** occur prior to or during any exit from

[14] As specified in SP 800-38C.

the routine – whether exiting because of an error, or exiting normally, with the output of *WrappedKM*.

7.1.2 Key-Unwrapping using AES-CCM

When party V receives *WrappedKM* and *OtherKWAInput*, the plaintext keying material *KM* may be recovered from *WrappedKM* using the key-wrapping key *KWK*; the received or agreed-upon MAC-tag length, *TLen*; the received nonce *Nonce*; and the received and/or previously known portions of any additional input *A* in the decryption-verification process for the CCM mode of AES. The unwrapping operation recovers the keying material *KM* from *WrappedKM* (the encrypted keying material, concatenated with a MAC tag) using the key-wrapping key *KWK*, *Nonce* and *A*, then verifies the integrity of *KM* and *A* by using the *KWK*, the *Nonce*, and the MAC tag.

Restrictions on the nonce *Nonce*;; the ciphertext *WrappedKM*; the additional input *A*; and the MAC-tag length, *TLen***,** are provided in SP 800-38C.

Function: KWA.UNWRAP(*KWK*, *WrappedKM*, *OtherKWAInput*)
Input:

1. *KWK*: The key-wrapping key; a 128-, 192- or 256-bit string.
2. *WrappedKM*: The ciphertext to be unwrapped; a bit string.
3. *OtherKWAInput*:
   a) *Nonce*: A nonce, as specified in Section 5.4; a bit string.
   b) *TLen*: The bit length of the MAC tag to be generated; an integer.
   c) *A*: The additional input (see Section 5.8.2); a byte string.

Process:

1. Check that the following conditions are satisfied:
   - The length of the *KWK* is the agreed-upon length (see assumption 2),
   - The value of *TLen* is valid for AES-CCM[15],
   - *KM* is valid for AES-CCM,
   - *Nonce* is valid for AES-CCM, and
   - *A* is valid for AES-CCM[20].

   If any of these conditions is not satisfied, return an error indicator, and exit without further processing.
2. (*status*, *KM*) = **CCM.Decrypt**(*KWK*, *TLen*, *Nonce, A*, *WrappedKM*).
3. If (*status* indicates an error), return *status*, and exit without further processing.

[15] The validity of *TLen*, *KM* and *Nonce* are discussed in Section 5.4 of SP 800-38C.

4. Return *KM*.

Output:

The plaintext keying material *KM* (a bit string), or an error indicator.

Note that the inputs to the **CCM.Decrypt** operation in process step 2 do not exactly match the specification of the Decryption-Verification process in SP 800-38C, in which (the equivalents of) *KWK* and *TLen* are listed as prerequisites, while the nonce, the additional input and *WrappedKM* are listed as inputs.

A routine that implements this operation **shall** destroy any local copies of sensitive input values (including *KWK* and any sensitive portions of *A*), any locally stored portions of *KM*, and any other potentially sensitive locally stored values used or produced during its execution. (The **CCM.Decrypt** routine **should** do the same.) Their destruction **shall** occur prior to or during any exit from the routine – whether exiting early, because of an error, or exiting normally, with the output of *KM*. Note that the requirement for destruction includes any locally stored portions of the unwrapped (i.e., plaintext) keying material *KM*.

7.1.3 Key Wrapping Using KW or KWP

The KW and KWP modes of AES used for key wrapping do not include methods for handling additional input; therefore, these methods **shall not** be used when additional input needs to be included with the keying material *KM* (i.e., the *OtherKWAInput* parameter is not used).

The keying material to be wrapped[16], *KM*, is input to the KW or KWP modes of AES specified in SP 800-38F. The wrapping operation encrypts and integrity protects the keying material using a key-wrapping key[17] *KWK*. Limitations on the length of *KM* are provided in Section 5.3.1 of SP 800-38F.

Function: KWA.WRAP(*KWK*, *KM*)

Input:

1. *KWK*: The key-wrapping key.
2. *KM*: The keying material to be wrapped; a semi-block string for KW, or a byte string for KWP (see SP 800-38F for details).

Process:

1. If the length of *KM* is not valid, then return an error indicator and exit without further processing.
2. *WrappedKM* = **Wrap**(*KWK*, *KM*).
3. Return *WrappedKM*.

Output: Ciphertext *WrappedKM*.

In process step 2, **Wrap** is either **KW-AE** or **KWP-AE**, as specified in SP 800-38F.

[16] Called the plaintext *P* in SP 800-38F.

[17] Called *K* in SP 800-38C.

Also, note that the inputs to the **Wrap** operation in step 2 do not exactly match the specification for the KW and KWP wrapping methods in SP 800-38F, in which *KWK* is listed as a prerequisite, while *KM* is listed as an input.

A routine that implements this operation **shall** destroy any local copies of the input values *KWK* and *KM*, as well as any other potentially sensitive locally stored values used or produced during its execution. (The **Wrap** routine **should** do the same.) Their destruction **shall** occur prior to or during any exit from the routine – whether exiting because of an error, or exiting normally, with the output of *WrappedKM*.

7.1.4 Key Unwrapping Using KW or KWP

The unwrapping operation recovers the keying material *KM* from the ciphertext *WrappedKM* using the key-wrapping key *KWK*. Limitations on the length of *WrappedKM* are provided in Section 5.3.1 of SP 800-38F.

Function: KWA.UNWRAP(*KWK*, *WrappedKM*)

Input:

1. *KWK*: The key-wrapping key.
2. *WrappedKM*: The ciphertext to be unwrapped; a byte string.

Process:

1. If the length of *WrappedKM* is not valid, then return an error indicator, and exit without further processing.
2. (*status*, *KM*) = **Unwrap**(*KWK*, *WrappedKM*).
3. If (*status* indicates an error), return *status*, and exit without further processing.
4. Return *KM*.

Output:

The plaintext keying material *KM*, or an indication of an error.

In process step 2, **Unwrap** is either **KW-AD** or **KWP-AD**, as specified in SP 800-38F.

Note that in process step 2, the returned values have been slightly altered from those specified in SP 800-38F. In SP 800-38F, either the plaintext keying material or a "FAIL" indicator is returned, whereas process step 2 is specified with two return values: an indication of the status of the operation (e.g., SUCCESS or FAIL) and the plaintext keying material if the **Unwrap** operation does not indicate "FAIL.".

In addition, the inputs to the **Unwrap** operation in process step 2 do not exactly match the specification in SP 800-38F, in which *KWK* is listed as a prerequisite, while *WrappedKM* is listed as an input.

A routine that implements this operation **shall** destroy any local copies of the input value *KWK*, any locally stored portions of *KM*, and any other potentially sensitive locally stored values used or produced during its execution (the **Unwrap** routine **should** do the same.) Their destruction **shall** occur prior to or during any exit from the routine – whether exiting early because of an error, or exiting normally, with the output of *KM*. Note that the requirement for destruction includes any locally stored portions of the unwrapped (i.e., plaintext) keying material *KM*.

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 keying material *KM* to be wrapped and 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 transported keying material *KM* for each key-transport receiver that is expected to provide key confirmation to party U.

The minimum lengths of the *MacKey* and the *MacTag* **shall** be selected as specified in Section 5.9.3.

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

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

The *KeyData* may be *Null*, or may contain keying material to be used after 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 *KM* 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 length) on the *MacData* (see Section 5.9.1.1) represented as

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

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. *WrappedKM* 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 *KM*, 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 transported keying material *KM* and the *MacData* formed as described above). Party U then verifies that this newly computed *MacTag* matches the *MacTag* value provided by party V.

7. DLC-Based Key Transport (Alternative 2)

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 (*KWK*) **shall** be established using either a C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) key-agreement scheme; a C(2e, 0s) scheme **shall not** be used to establish the key-wrapping key.

KWK **shall** then be used by party U to wrap secret keying material using an **approved** key-wrapping algorithm, based on the use of AES. Three methods of key wrapping are **approved** for DLC-based key transport: CCM, KW and KWP; CCM is specified in SP 800-38C, while KW and KWP are specified in SP 800-38F. Note that for DLC-based key transport, party U in the key-agreement scheme is the key-transport sender, and party V is the receiver.

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.

7.1 Assumptions

In order to execute a DLC-based key-transport scheme in compliance with this Recommendation, the following assumptions **shall** be true:

1. All assumptions for the key-agreement scheme used **shall** be true (see Sections 6.1.1, 6.2.1, 6.2.2 and 6.3).
2. The sender and receiver have agreed upon an **approved** AES variant (i.e., AES-128, AES-192 or AES-256) and key-wrapping method (i.e., either CCM, KW or KWP). The key-wrapping method **shall** protect the transported keying material at a security strength that is equal to or greater than the target security strength of the applicable key-establishment scheme.

   If the CCM mode is used during key wrapping, the sender and receiver have agreed on the counter-generation function, the formatting function, and *TLen*, the bit length of the CBC-MAC tag to be produced during the key-wrapping operation.

   If the KW or KWP mode is used for key wrapping, the sender and receiver have agreed on the valid plaintext lengths to be used during key wrapping.
3. If the CCM mode will be used for key wrapping, prior to or during the key-establishment process, the parties have either agreed upon the format and content of the additional input *A* (a string to be cryptographically bound to the transported keying material so that the cipher is a cryptographic function of both values), or agreed that *A* will be the empty string. Note that for the KW and KWP modes, additional input is not accommodated.
4. If the CCM mode is used for key wrapping, either party U and party V **shall** have agreed on the MAC-tag length (*Tlen*) for the key-wrapping process, or party U **shall** send the CBC-MAC-tag length to party V, along with the wrapped keying material.

5. The sender and receiver have agreed on whether or not key confirmation will be used following the transport of the wrapped keying material. If key confirmation is used, the parties have also agreed upon an **approved** MAC algorithm and associated parameters, including the lengths of *MacKey* and *MacTag*, as specified in Section 5.9.3).

6. Prior to or during the key-agreement process, the keying material to be transported has been (or will be) determined.

7.2 Key-Transport Scheme

The DLC-based key transport scheme is as follows:

1. A key agreement scheme is used between party U and party V to establish a shared secret and derive keying material that includes *KWK*. Key confirmation (as specified in Sections 5.9 and 6) may optionally be performed.

2. The sender (party U) selects secret keying material, *KM*, to transport to the receiver (party V). If key confirmation is to be performed following key transport, *KM* **shall** include a fresh (i.e., not previously used) *MacKey* to be used for key confirmation and the *KeyData* to be used subsequent to key transport.

4. The sender calculates *WrappedKey* = KeyWrap(*KWK*, *KM*, *other_inputs*), where *other_inputs* are any additional inputs needed for the selected, **approved** key-wrapping algorithm KeyWrap( ).

5. The sender sends *WrappedKey* to the receiver.

6. The receiver receives *WrappedKey* from the sender.

7. The receiver obtains *KWK* from the derived keying material that is computed by applying the key derivation function to the shared secret.

8. The receiver calculates *KM* = KeyUnwrap(*KWK*, *WrappedKey*, *other_inputs*), where *other_inputs* are any additional inputs needed for the appropriate **approved** key-unwrapping algorithm KeyUnwrap( ).

9. If key confirmation is to be performed following key transport, then both the sender and receiver **shall** proceed as specified in Section 7.3.

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

A default "rule" of this Recommendation is that ephemeral keys **shall not** be reused (see Section 5.6.3.3). An exception to this rule is that the sender may use the same ephemeral key pair in step 1 above in multiple DLC-based key-transport transactions if the same secret keying material is being transported in each transaction and if all these transactions 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.

7.3 Key Confirmation for Transported Keying Material

If key confirmation is to be provided in compliance with this Recommendation following the transport of keying material, party U **shall** generate a fresh *MacKey* and include it as part of the keying material *KM* to be wrapped and 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 supports the security strength required for the key-establishment transaction. In broadcast scenarios, a different *MacKey* **shall** be included in the transported keying material *KM* for each key-transport receiver that is expected to provide key confirmation to party U.

The minimum lengths of the *MacKey* and the *MacTag* **shall** be selected as specified in Section 5.9.3.

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

$$KM = MacKey \parallel KeyData.$$

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 *KM* 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 length) on the *MacData* represented as

$$MacData = \text{"KC\_KT"} \parallel ID_V \parallel ID_U \parallel EphemData_V \parallel EphemData_U \parallel WrappedKM \{ \parallel Text\},$$

where ID_V is the identifier associated with party V (the receiver), and ID_U is the identifier associated with party U (the sender). 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. *WrappedKM* 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 *KM*, 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 transported keying material *KM* 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 present possible justifications for selecting schemes from each subcategory, C(*ie*, *j*s). The proffered rationales are intended to provide the user and/or developer with some information that may help when deciding key-agreement scheme to be used. The rationales include brief discussions of basic security properties, but do not constitute an in-depth analysis of all possible security properties of all schemes under all adversary models. The specific security properties that are cited will depend on such considerations as 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 into a scheme. In general, the security properties cited for a subcategory of schemes are exhibited by each scheme within that subcategory; 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 may possess. 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 exchange consumes a considerable amount of time, a scheme with fewer exchanges during a single key-agreement transaction might be preferable to a scheme with more exchanges, even though the latter may possess more security benefits. It is important to keep in mind that a key-agreement scheme may be a component of a larger protocol that offers additional security-related assurances beyond those provided by the key-agreement scheme alone. For example, the protocol may include specific features that limit opportunities for accidental or intentional misuse of the key-agreement component of the protocol. Protocols, per se, are not specified in this Recommendation.

Important Note: In order to provide concise descriptions of security properties possessed by the various schemes, it is necessary to make some assumptions concerning the format and type of data that is used as input 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 security properties of a subcategory of schemes, it is assumed that the *FixedInfo* input to a key-derivation method employed during a particular key-agreement transaction uses either the concatenation format or the ASN.1 format (see Sections 5.8.2.1 and 5.8.2.2). It is also assumed that *FixedInfo* includes sufficiently specific identifiers for the participants in the transaction, an identifier for the key-agreement scheme being used during the transaction, and additional input (e.g., a nonce, ephemeral public key, and/or session identifier) that may provide assurance to one or both participants that the derived keying material will reflect the specific context in which the transaction occurs (see Section 5.8.2 and Appendix B for further discussion concerning context-specific information that may be appropriate for inclusion in *FixedInfo*). In general, *FixedInfo* may include pre-shared secrets, but that is not assumed to be the case in the analysis of security properties that follows. In cases where an **approved** extraction-then-expansion key-derivation procedure is employed (see SP 800-56C), it is assumed that this

FixedInfo is used as the *Context* input during the key-expansion step. 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

These schemes require each participant to own a static key pair that is used in their key-agreement transaction. Static key pairs can provide the participants with some level of assurance that they have correctly identified the party with whom they will be establishing keying material if the transaction is successfully completed.

In the case of a key-agreement transaction based on the Full Unified model or dhHybrid1 scheme, each participant has assurance that no unintended entity (i.e., no entity other than the owners of the static key pairs involved in the transaction) could employ a Diffie-Hellman primitive (see Section 5.7.1) to compute Z_s, the static component of the shared secret Z without knowledge of one of the static private keys employed during the transaction. Absent the compromise of Z_s or one of those static private keys, each participant can be confident of correctly identifying the other participant in the key-establishment transaction. The level of confidence is commensurate with the specificity of the identifiers that are associated with the static public keys (and are used as input during the key-derivation process), the degree of trust in the association between those identifiers and static public keys, the assurance of validity of the domain parameters and static public keys, and the availability of evidence that the keying material has been correctly derived.

Similarly, in the case of a key-agreement transaction based on Full MQV or MQV2, each participant has assurance that no unintended entity could use a DLC primitive to compute the shared secret Z without knowledge of either a static private key or a private-key-dependent implicit signature employed during the transaction. (The term "implicit signature" refers to those quantities denoted S_A and $implicitsig_A$ in the descriptions of the MQV primitives in Section 5.7.2.1 and Section 5.7.2.3, respectively.) Absent the compromise of Z, a static private key, or an implicit signature, each participant can be confident of correctly identifying the other participant in the key-establishment transaction. As above, the level of confidence is commensurate with the specificity of the identifiers that are associated with the static public keys (and are used as input during the key-derivation process), the degree of trust in the association between those identifiers and static public keys, the assurance of validity of the domain parameters and static public keys, and the availability of evidence that the keying material has been correctly derived.

These schemes also require each participant to generate an ephemeral key pair that is used in their transaction, providing each participant with assurance that the resulting shared secret (and the keying material derived from it) will vary from one of their C(2e, 2s) transactions to the next.

Each participant in a C(2e, 2s) transaction has assurance that the value of the resulting shared secret Z will not be completely revealed to an adversary who is able to compromise (only) their static private keys at some time after the transaction is completed. (The adversary would, however, be able to compute Z_s, the static component of the shared secret, if the key-agreement transaction was based on the Full Unified model or dhHybrid1 scheme.) This assurance is commensurate with the confidence that a participant has that neither of the ephemeral private keys employed in the transaction will be compromised. By generating their ephemeral key pairs as close to the time of use as possible and destroying the ephemeral private keys after their use, the participants reduce the risk of such a compromise.

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 Full MQV) will be employed during a transaction with an adversary who is in possession of a compromised static private key (or a compromised implicit signature corresponding to that static private key), the adversary is limited to masquerading as the owner of the compromised key pair (or as the owner of the static key pair corresponding to the compromised implicit signature). 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 any other entity during a Full Unified model- or dhHybrid1-based key-agreement transaction with that entity. Also, the compromise of Z_s, the static component of a shared secret that was (or would be) 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 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 a C(2e, 2s) scheme 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 some compromise of secret information (e.g., a static private key or a static component of *Z*), a key-confirmation recipient can obtain assurance that the appropriate 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.

8.2 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 that involves two honest parties and is not influenced by an adversary) is protected against any compromise of shared secrets and/or private keys associated with 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, by themselves, 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 (e.g., in the protocol incorporating the key-agreement scheme).

This Recommendation does not specify the incorporation of key confirmation in a C(2e, 0s) scheme.

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, the participant acting as party U is required to generate and use an ephemeral key pair. Different assurances are provided to the participants by the utilization of a C(1e, 2s) scheme, depending upon which one acts as party U and which one acts as party V.

The use of static key pairs in the key-agreement transaction can provide the participants with some level of assurance that they have correctly identified the party with whom they will be establishing keying material if the transaction is successfully completed.

In the case of a transaction based on the One-Pass Unified model or dhHybridOneflow scheme, each participant has assurance that no unintended entity (i.e., no entity other than the owners of the static key pairs involved in the key-establishment transaction) could employ a Diffie-Hellman primitive (see Section 5.7.1) to compute Z_s, the static component of the shared secret Z, without knowledge of one of the static private keys employed during the transaction. Absent the compromise of Z_s or one of those static private keys, each participant can be confident of correctly identifying the other participant in the key-establishment transaction. The level of confidence is commensurate with the specificity of the identifiers that are associated with the static public keys (and are used as input during the key-derivation process), the degree of trust in the association between those identifiers and static public keys, the assurance of validity of the domain parameters and static public keys, and the availability of evidence that the keying material has been correctly derived.

Similarly, in the case of a key-agreement transaction based on the One-Pass MQV or MQV1 scheme, each participant has assurance that no unintended entity could use a DLC primitive to compute the shared secret Z without knowledge of either the static private key of one of the participants in the transaction or the private-key dependent *implicit signature* employed by party U during the transaction. (The term "implicit signature" refers to those quantities denoted S_A and $implicitsig_A$ in the descriptions of the MQV primitives in Section 5.7.2.1 and Section 5.7.2.3, respectively.) Absent the compromise of Z, a static private key, or party U's implicit signature, each participant can be confident of correctly identifying the other participant in the key-establishment transaction. As above, the level of confidence is commensurate with the specificity of the identifiers that are associated with the static public keys (and are used as input during the key-derivation process), the degree of trust in the association between those identifiers and static public keys, the assurance of validity of the domain parameters and static public keys, and the availability of evidence that the keying material has been correctly derived.

Party U, whose ephemeral key pair is used in the computations, has assurance that the resulting shared secret will vary from one C(1e, 2s) transaction to the next such transaction with the same party V. The participant acting as 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 in a key-confirmation process performed as specified in this Recommendation). The assurance of freshness of the derived keying material that can be obtained in this way by the participant acting as party V is commensurate with the participant's assurance that a different nonce will be contributed during each such transaction.

The compromise of the static private key used by party U does not, by itself, compromise the shared secret computed during any legitimate C(1e, 2s) transaction (i.e., a transaction involving two honest parties). Likewise, the compromise of only the ephemeral private key used by party U 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 (regardless of the static or ephemeral keys used by the entity acting as party U); to compromise those 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 particular entity, the adversary may be able to impersonate any other entity while acting as party U in a C(1e, 2s) transaction in which the owner of the compromised static private key acts as party V. Similarly, the compromise of the static component, Z_s, 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. If the MQV1 or One-Pass MQV scheme will be employed during a key-agreement transaction with an adversary who is in possession of a compromised implicit signature corresponding to a static private key, the adversary may be able to masquerade as the owner of that static key pair while acting as party U (provided that the static key pair is compatible with the domain parameters employed during the transaction).

Key confirmation can be provided in either or both directions as part of a C(1e, 2s) scheme 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 secret information (e.g., a static private key or a static component of *Z*), a key-confirmation recipient can obtain assurance that the appropriate 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.

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

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

The use of a static public key attributed to party V can provide the participant acting as party U with some level of assurance that he has correctly identified the party with whom he will be establishing keying material if the transaction is successfully completed.

Whether the transaction is based on the One-Pass Diffie-Hellman or dhOneflow scheme, the participant acting as party U has assurance that no unintended entity (i.e., no entity other than himself and the owner of the static public key attributed to party V) could employ a Diffie-Hellman primitive (see Section 5.7.1) to compute the shared secret *Z* without knowledge of one of the private keys employed during the transaction. Absent the compromise of *Z* or one of those private keys, the participant acting as party U can be confident of correctly identifying the other participant in the key-establishment transaction as the owner of the static public key attributed to party V. The level of confidence is commensurate with the specificity of the identifier that is associated with the static public key attributed to party V (and is used as input during the key-derivation process), the degree of trust in the association between that identifier and the static public key, the assurance of validity of the domain parameters and static public key, and the availability of evidence that the keying material has been correctly derived.

The participant acting as party V has no such assurance, in general, since he 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 party U contributes to the transaction).

The participant acting as party U, whose ephemeral key pair is used in the computations, has assurance that the resulting shared secret will vary from one C(1e, 1s) transaction to the next. The participant acting as 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 participant that the security of the shared secret is protected against the compromise of a private key. A compromise of the ephemeral private key used in a C(1e, 1s) transaction only compromises the shared secret resulting from that particular transaction (and by generating the ephemeral key pair as close to the time of use as possible and destroying the ephemeral private key after its use, the participant acting as party U reduces the risk of such a compromise). However, the compromise of an entity'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 (no matter what party plays the role of party U); to compromise those shared secrets, the adversary must also acquire the ephemeral public keys contributed by whomever acts as party U in those transactions. In addition, if an adversary learns a particular entity's static private key, the adversary may be able to impersonate that particular entity while acting as party V in a C(1e, 1s) transaction that employs compatible domain parameters.

The participant acting as party V may provide key confirmation to party U as specified in Section 6.2.2.3. This allows the participant acting as party U (who is the key confirmation recipient) 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 secret information (e.g., a private key), the participant acting as party U can obtain assurance that the appropriate identifier has been used to label party V, and that the participant acting as party V is indeed the owner of the static public key associated with that identifier. Under such circumstances, the participant acting as party U can also receive assurance of the active (and successful) participation in the key-agreement transaction by the owner of the static public key attributed to party V.

This Recommendation does not specify the incorporation of key confirmation from party U to party V in a C(1e, 1s) scheme.

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, the participant acting as party U is required to generate a nonce, which is sent to party V and used (by both participants) as input to their chosen key-derivation method.

The use of static key pairs in the key-agreement transaction can provide the participants with some level of assurance that they have correctly identified the party with whom they will be establishing keying material if the transaction is successfully completed.

Whether the transaction is based on the Static Unified Model or dhStatic scheme, each participant has assurance that no unintended entity (i.e., no entity other than the owners of the static key pairs employed in the transaction) could employ a Diffie-Hellman primitive (see Section 5.7.1) to compute the static shared secret *Z* without knowledge of one of the static private keys employed during the transaction. Absent the compromise of *Z* or one of those static private keys, each participant can be confident of correctly identifying the other party in the key-establishment transaction. The level of confidence is commensurate with the specificity of the identifiers that are associated with the static public keys (and are used as input during the key-derivation process), the degree of trust in the association between those identifiers and static public keys, the assurance of validity of the domain parameters and static public keys, and the availability of evidence that the keying material has been correctly derived.

Although the value of *Z* is the same in all C(0e, 2s) key-establishment transactions between the same two parties (as long as the two participants employ the same static key pairs), the participant acting as party U, whose (required) nonce is used in the key-derivation computations, has assurance that the derived keying material will vary from one of their C(0e, 2s) transactions to the next. In general, the participant acting as 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). The assurance of freshness of the derived keying material that can be obtained by a participant in a C(0e, 2s) transaction is commensurate with the participant's assurance that a different nonce will be contributed during each such transaction.

If the static *Z* value formed by the two participants 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 (at least) the nonce contributed by the participant that acted as party U in that transaction. The compromise of the static *Z* value may also permit an adversary to masquerade as either entity to the other entity in future C(0e, 2s) key-agreement transactions.

If a particular entity's static private key is compromised, then shared secrets resulting from 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 when interacting with the compromised entity, 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 transaction, the adversary must acquire (at least) the static public key contributed by the other entity participating in that transaction with the compromised entity, 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 (if the compromised key pair is of the type permitted by the scheme and domain parameters) the adversary may masquerade as the owner of the compromised static key pair in key-agreement transactions with any other party. In addition, the adversary may 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 a C(0e, 2s) scheme 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 (e.g., a static private key or the static shared secret, *Z*), a key-confirmation recipient can obtain assurance that the appropriate 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 which the intended key-transport sender acts as party U, and the intended key-transport receiver acts as party V. The basic security properties of these schemes have been described in the previous sections. The following discussion emphasizes the effects that the properties of the key-agreement scheme used to establish a key-wrapping key may have on assurances that can be provided to the sender and/or receiver of the wrapped keying material.

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

Each of the schemes that can be used during the key-agreement phase of the transaction requires the use of a static public key owned by the participant acting as party V. Unless there is a compromise of some secret information (e.g., a static component of *Z* or a private key), the key-transport sender (who acts as party U) has assurance that no unintended entity (i.e., no parties other than himself and the owner of the static public key attributed to party V) could employ a DLC primitive to compute the shared secret *Z* that is used to derive the key-wrapping key used during the key-transport process. Absent such a compromise, the key-transport sender can be confident that he has correctly identified the key transport receiver (assumed to have been acting as party V). The level of confidence is commensurate with the specificity of the identifier that is associated with the static public key attributed to party V, the degree of trust in the association between that identifier and that static public key, the assurance of validity of the domain parameters and public keys employed during the key-agreement phase of the transaction, and the availability of evidence that the key-wrapping key has been correctly derived by the key-transport receiver.

When a C(2e, 2s), C(1e, 2s), or C(1e, 1s) scheme is employed during the key-agreement portion of the transaction, the key-transport sender (i.e., party U) generates an ephemeral key pair that is used in the computation of *Z*. This provides assurance to party U (the key-transport sender) that both the shared secret and the derived key-wrapping key will vary from one key-transport transaction to the next. Assurance of the freshness of the derived key-wrapping key may also be obtained by party U when a C(0e, 2s) scheme is employed. In that case, party U is required to contribute a nonce (see Section 5.4) that is used in the derivation of the key-wrapping key; the assurance of freshness that party U (the key-transport sender) can obtain is commensurate with the probability that the contributed nonce has not been previously employed in the key-derivation process of the key-agreement portion of some other transaction. Assurance that a fresh key-wrapping key is used during each instance of key transport provides commensurate assurance to party U (the key-transport sender) that the confidentiality of the wrapped keying material transported during a transaction with party V will not be threatened by the possibility that the key-wrapping key has been (or will be) compromised as a result of its use in some other transaction and/or application.

Assuming that no key pairs and/or static *Z* values are compromised, the required use of a static public key attributed to party V (the intended key-transport receiver) during the key-agreement portion of the transaction, together with each scheme's required ephemeral contribution from party U, provides assurance to party U (the key-transport sender) that the owner of the static private key attributed to party V is the only other party who will be able to acquire 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 during the key-agreement portion of the transaction, the use of a static public key attributed to party U (the key-transport sender) provides the participant acting as party V (the key-transport receiver) with a means of identifying the entity with whom he will be establishing keying material if the transaction is successfully completed. The trusted association of an identifier with a static public key attributed to party U provides party V with a method for accurately labeling the (purported) key-transport sender (i.e., party U). Absent the compromise of some secret information (e.g., a static component of *Z* or a private key), party V can be confident that no unintended entity (i.e., no parties other than himself and the owner of the static public key attributed to party U) could employ a DLC primitive to compute the shared secret *Z*, from which the key-wrapping key is derived. Party V's confidence is commensurate with the specificity of the identifier that is associated with the static public key attributed to party U, the degree of trust in the association between that identifier and that static public key, the assurance of validity of the domain parameters and public keys employed during the transaction, and the evidence available to party V that party U has derived the correct key-wrapping key (i.e., the key used by party U to wrap the transported keying material).

On the other hand, if a C(1e, 1s) scheme is employed during the key-agreement portion of the transaction, 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, 1s) scheme (in and of itself) offers no assurance to the party V (the key-transport receiver) of the accuracy of any identifier that may be associated with party U. Any trusted association desired/required between an identifier and the (purported) key-transport sender (party U) would have to be provided by methods external to the key-establishment scheme.

When a C(2e, 2s) scheme is employed during the key-agreement portion of the transaction, the key-transport receiver (acting as 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 and party V contributes a nonce (see Section 5.4) that is used in the derivation of the key-wrapping key. The assurance of freshness that party V can obtain in this way is commensurate with the probability that the contributed nonce has not been previously employed in a key-derivation process. Assurance that a fresh key-wrapping key is used during each instance of a key-transport transaction provides commensurate assurance to party V that the confidentiality of the wrapped keying material transported during a transaction with party U will not be threatened by the possibility that the key-wrapping key has been (or will be) compromised as a result of the use of an identical key in some other transaction and/or application.

Key confirmation from party V (the intended key-transport receiver) to party U (the intended key-transport sender) can be incorporated into a C(2e, 2s), C(1e, 2s), C(1e, 1s) or C(0e, 2s) key-agreement 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 U (the intended key-transport sender) to obtain assurance that party V (the intended key-transport receiver) has derived the correct key-wrapping key. A key-confirmation failure 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 into a C(2e, 2s), C(1e, 2s), or C(0e, 2s) key-agreement 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-wrapping key is derived. Key confirmation provided in this direction (from party U to party V) enables party V to obtain assurance that he has derived the same key that party U will employ to wrap the transported keying material. A key-confirmation failure 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 (the key-transport sender) to obtain assurance that party V (the intended key-transport receiver) 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 portion of the transaction.

The use of a C(1e, 2s), C(1e, 1s) or C(0e, 2s) key-agreement scheme to establish the key-wrapping key allows for one-pass implementations of key transport (in cases where key confirmation is not required). If the static public key attributed to party V (the intended key-transport receiver) has been obtained previously, party U (the key-transport sender) 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 key-agreement portion of the transaction.

There are additional considerations that apply to the broadcast scenario, in which one sender (acting as party U) transports the same keying material "simultaneously" (or within a short period of time) to multiple receivers (i.e., multiple entities acting as party V) 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, the parties must proceed with caution when engaging in such practices (e.g., see "On Reusing Ephemeral Keys in Diffie-Hellman Key Agreement Protocols," by A. Menezes and B. Ustaoglu, which is available at the following url: http://cacr.uwaterloo.ca/techreports/2008/cacr2008-24.pdf).

As part of the proper implementation of this Recommendation, the key-transport sender (acting as party U) **should not** reuse an ephemeral public key when establishing key-wrapping keys for key transport in a broadcast scenario unless all parties involved 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 the sender and the various receivers.

If, in a broadcast scenario, the key-transport sender (i.e., party U) requires multiple key-transport 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 the sender to transport a different 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 the 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 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 to claim conformance to this Recommendation. Information on the CAVP and 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 or selection 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 Sections 5.6.2.3.1 and 5.6.2.3.2 for FFC or as specified in Sections 5.6.2.3.3 or 5.6.2.3.4 for ECC.
- A key-agreement scheme from Section 6, together with an **approved** key-derivation method from SP 800-56C. 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,
- The domain parameter generation method or the selected domain parameters (see Section 5.5.1),
- The hash function(s) used, if appropriate (see Section 5.1),
- The MAC algorithm(s) used, if appropriate (see Section 5.2),
- The MAC key length(s) (see Section 5.9.3),
- The MAC tag length(s) (see Section 5.9.3).
- The type of cryptography: FFC or ECC,
- The key-establishment schemes available (see Sections 6 and 7),
- The key-derivation method to be used, including the format of *FixedInfo* (see Section 5.8 and SP 800-56C),
- The type of nonces to be generated (see Section 5.4),

- The NIST-Recommended elliptic curve(s) available (if appropriate),
- The key-wrap algorithm used for key transport (see Section 7), if appropriate, and
- The key-confirmation scheme, if appropriate (see Section 5.9).

Appendix A: References

A.1 Normative References

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

[FIPS 140 Annex A]

Approved Security Functions, Draft, April 2016.

[FIPS 140 Annex D]

Approved Key Establishment Techniques. Draft, October 2014.

[FIPS 140 IG]

Federal Information Processing Standard 140-2 Implementation Guidance, Available at http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf .

[FIPS 180] Federal Information Processing Standard 180-4, Secure Hash Standard, August, 2015.

[FIPS 186] Federal Information Processing Standard 186-4, Digital Signature Standard, July 2013.

[FIPS 197] Federal Information Processing Standard 197, Advanced Encryption Standard, November 2001.

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

[FIPS 202] Federal Information Processing Standard 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015.

[SP 800-38B] Special Publication 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, May 2005, with updates dated October 2016.

[SP 800-38C] Special Publication 800-38C, Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality, May 2004, with updates dated July 2007.

[SP 800-38F] Special Publication 800-38F, Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, December, 2012.

[SP 800-52] Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations, April 2014.

[SP 800-56B] Special Publication 800-56B, Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography, Revision 1, September 2014.

[SP 800-56C] Special Publication 800-56C, Recommendation for Key Derivation Methods in Key Establishment Schemes (DRAFT), November 2017.

[SP 800-57] Special Publication 800-57, Part 1: Recommendation for Key Management, Revision 4, January 2016.

[SP 800-67] Special Publication 800-67, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, Revision 1, January 2012.

[SP 800-90] Special Publication 800-90 series:

Special Publication 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Revision 1, June 2015.

Special Publication 800-90B, DRAFT Recommendation for the Entropy Sources Used for Random Bit Generation, January 2016.

Special Publication 800-90C, DRAFT Recommendation for Random Bit Generator (RBG) Constructions, April 2016.

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

[SP 800-131A] Special Publication 800-131A, Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, Revision 1, November 2015.

[SP 800-133] Special Publication 800-133, *Recommendation for Cryptogrsphic Key Generation*, December 2012.

[SP 800-135] Special Publication 800-135, Recommendation for Existing Application-Specific Key Derivation Functions, Revision 1, December 2011.

SP 800-185] Special Publication 800-185, SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash, December 2016.

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

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

[ANS X9.63] American National Standard 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.

[RFC 3526] More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE), May 2003, see https://www.ietf.org/rfc/rfc3526.txt.

[RFC 4492] Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS), May 2006, see https://www.ietf.org/rfc/rfc4492.txt.

[RFC 5903] Elliptic Curve Groups Modulo a Prime (ECP Groups) for IKE and IKEv2, June 2010, see https://tools.ietf.org/html/rfc5903.

[RFC 7919] Negotiated Finite Field Diffie-Hellman Ephemeral Parameters, August 2016, see https://tools.ietf.org/html/rfc7919.

[SECG] Standards for Efficient Cryptography Group, see http://www.secg.org/.

[SEC2] Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, September 2000, see http://www.secg.org/SEC2-Ver-1.0.pdf.

A.2 Informative References

[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.

[ISO/IEC 8825-1]

Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 2008.

[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 and other Context-specific Information in the KDM Input (Informative)

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 $t = \lceil \log_2 p \rceil$ and p is the large prime field order.

Output: A byte string S of length n bytes.

1. $J_{n+1} = C$.
2. For $i = n$ to 1 by -1

   2.1 $J_i = \lfloor (J_{i+1})/256 \rfloor$.

   2.2 $A_i = J_{i+1} - (J_i \bullet 256)$.

   2.3 $S_i = (a_{i1}, a_{i2}, a_{i3}, a_{i4}, a_{i5}, a_{i6}, a_{i7}, a_{i8})$,
   The 8-bit binary representation of the non-negative integer
   $A_i = a_{i1}\, 2^7 + a_{i2}\, 2^6 + a_{i3}\, 2^5 + a_{i4}\, 2^4 + a_{i5}\, 2^3 + a_{i6}\, 2^2 + a_{i7}\, 2 + a_{i8}$.

3. Let $S_1, S_2, \ldots, S_n$ be the bytes of S from leftmost to rightmost.
4. Output S.

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 it is assumed that α is (already) represented as a bit string of length m, with each bit indicating the coefficient (0 or 1) of a specific element of a particular basis for $GF(2^m)$ viewed as a vector space over $GF(2)$.

   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 $(8n - m + 1)^{\text{th}}$ 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:

$$x = \sum 2^{(m-i)} s_i \qquad \text{for } i = 1 \text{ to } m.$$

C.4 Conversion of a Bit String to an Integer

An n-long sequence of bits $\{ x_1, \ldots, x_n \}$ is converted to an integer by the rule

$$\{ x_1, \ldots, x_n \} \rightarrow (x_1 * 2^{n-1}) + (x_2 * 2^{n-2}) + \ldots + (n_1 * 2) + x_n .$$

Note that the first bit of a sequence corresponds to the most significant bit of the corresponding integer, and the last bit corresponds to the least significant bit.

Input:

1. $b_1, b_2, \ldots, b_n$ The bit string to be converted.

Output:

1. C The requested integer representation of the bit string.

Process:

1. Let $(b_1, b_2, \ldots, b_n)$ be the bits of b from leftmost to rightmost.
2. $C = \sum_{i=1}^{n} 2^{(n-i)} b_i$.
3. Return C.

The binary length of an integer C is defined as the smallest integer n satisfying $C < 2^n$.

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 May 2013, 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 sender 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 party U and party 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 – 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 interval [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 interval [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 one-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 5.8.1 – Split Table 1 (for FFC) to Table 1 (Section 5.5.1.1), Table 6 (Section 5.8.1) and Table 8 (Section 5.9.3), where Table 1 is for FFC parameter-size sets, Table 6 is for the function H used for key derivation and Table 8 is about the MAC key length and MAC tag length. In the new tables, added row on “Maximum security strength supported”.
- Section 5.8.1 – In Table 6, changed the minimum output length for function H from 128 to 112 for FFC parameter set.
- Section 5.8.1 - Split Table 2 (for ECC) to Table 2 (Section 5.5.1.2), Table 7 (Section 5.8.1) and Table 9 (Section 5.9.3), where Table 2 is for ECC parameter-size sets, Table 7 is for the function H used for key derivation, and Table 9 is about the MAC key length and MAC tag length. In the new tables, added row on “Maximum security strength supported”.
- 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.
- New Section 5.9.3 – Table 8, changed the minimum *MacLen*, that is, *MacTag* length to 64 bits for all the parameter sets of FFC.
- New Section 5.9.3 – Table 9, changed the minimum *MacLen*, that is, *MacTag* length to 64 bits for all the parameter sets of ECC.
- Section 6 – The notation C(ie) replaces C(i), and C(ie, js) replaces C(i, j). If party 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.

In 2017, the following revisions were made:

1. Inserted hyperlinks for sections, references and definitions.
2. Tables 1, 2 6 and 7: Changed column 1, row 1 to "Targeted security strength" instead of "Maximum security strength supported"
3. Section 3.1: Added definitions for *critical security parameter* and *cryptographic module*. Updated the definition of *destroy, integrity, key-derivation procedure, key-establishment transaction, key wrapping, MacTagLen, message authentication code, shared secret symmetric key algorithm, store-and-forward* and *targeted security strength*. Modified the definition for *fresh, key confirmation, Mac tag* and *message authentication code.*

3. Section 3.2: Inserted *CSP,* len(x) and *RBG.* Removed *H* and *HMAC-hash*. Modified *MAC tag*.

4. Section 4: Inserted additional paragraphs the security of a key-establishment scheme and explicit instructions for the destruction of certain potentially sensitive values. Inserted a requirement that values explicitly required to be destroyed when leaving a routine (i.e., potentially sensitive locally stored data) **shall not** be used or reused for any additional purpose.
5. Section 4.1, paragraph 2, mentioned that domain prametrs may be from an approved list. Paragraph 3: Explained what is meant by transporting in a "protected manner."
6. Section 5.1: Inserted a reference to FIPS 202.
7. Section 5.2: Paragraph 2 – added KMAC to the list of approved MACs. Paragraph 3 – referred to SP 800-56C for the case where a MAC is used for key derivation. *MacLen* has been renamed to be *MacTagLen* for clarity.
8. Section 5.2.1, item 2: Changed “is required to” to “**shall**”. Added KMAC as a MAC algorithm.
9. Section 5.5: Revised wording.
10. Section 5.5.1.1: Certain FFC groups defined in other standards are now **approved** for use, which are encouraged for use. The old parameter-size sets in Table 1 are now addressed as FIPS 186-type sets and recommended for use only in legacy applications. Parameter-size set FA was removed. Table 1 has been shortened to address just the values of p and q; information about the hash function is now provided in Section 5.8.1 and Section 5.9.3. For the FIPS 186-type parameter-size sets, a requirement was added that the leftmost bit of p and q be a 1.

11. Section 5.5.1.2: Removed the table of parameter-size sets. Elliptic curves will be specified in SP 800-186 (when available; will continue to be available in FIPS 186 until then).
12. Section 5.5.2: Inserted an assurance method that allows **approved** safe-prime groups of domain parameters.
13. Section 5.6.1.1: Added discussions about the generation of key pairs for both the approved safe-prime groups and the FIPS 186-type parameter-size sets. The FFC key-pair generation routines from FIPS 186-4 were added (with some modifications). A reference to SP 800-133 is included for generating the keys.
14. Section 5.6.1.2: The ECC key-pair generation routines from FIPS 186-4 were added (with some modifications).
15. Section 5.6.2.1.2: Revised to accommodate the safe-prime groups.
16. Sections 5.6.2.1.3, 5.6.2.1.4 and 5.6.2.1.5: Revised for further clarity.
17. Section 5.6.2.1.4: The alternative test in method b was removed.
18. Section 5.6.2.2.2: Revised to accommodate the safe-prime groups.
19. Section 5.6.2.3: Introductory text added.
20. Section 5.6.2.3.1: Now specified as a method for FFC full public-key validation. The comment on process step 1 has been revised for clarity.
21. Section 5.6.2.3.2: New section added on FFC partial public-key validation.
22. Sections 5.6.2.3.1, 5.6.2.3.2 and 5.6.2.3.3: Added text to say that when an error is found, the routine should be exited immediately without further processing.
23. Section 5.6.2.2.2: Changed "The recipient of another party's ephemeral public key is required to obtain assurance…" to "The recipient of another party's ephemeral public key **shall** obtain assurance…".
24. Section 5.6.2.2.4, items 2 and 3: Added further clarifications.
25. Section 5.6.3.2: Public keys generated using the approved safe primes **shall not** be used for digital signatures.
26. Section 5.6.3.3: Added further clarification to item 1 to state that the private key needs to be protected until destroyed and is not to be backed up, archived or escrowed.
27. Section 5.7.1.1: Clarified error handling in step 2, and added checks for $z = p - 1$ and $z = 0$.
28. Section 5.7.1.2: Clarified error handling in step 2.
29. Section 5.7.2.1: Clarified error handling in step 6.
30. Section 5.7.2.3: Clarified error handling in step 3.
31. Section 5.8: Inserted a requirement that he shared secret **shall** be used only by an **approved** key-derivation method and **shall not** for any other purpose. Inserted an

explicit statement that SP800-56A approves the key-derivation methods only for the derivation of keys from a shared secret.

Moved all key-derivation methods to SP 800-56C. Inserted a new section (Section 5.8.1) to describe how to call a key-derivation method and reorganized Section 5.8.

To avoid confusion between the use of *OtherInput* and *OtherInfo* in the previous version of this document, *OtherInfo* was changed to *FixedInfo*; this information is used as fixed input to the key-derivation method. *keydatalen* was changed to *L* for (eventual) consistency between SP 800-56A/B/C and SP 800-108.

32. In the new Section 5.8.2.1, inserted text in *SuppPubInfo* and *SuppPrivInfo* that states that, while an implementation may be capable of including these subfields, the subfields may be null for a given transaction.
33. Section 5.8.2.2 clarifies the interaction with the two-step key-derivation procedure in SP 800-56C.
34. Section 5.9.1: Changed “Each party is required to have an identifier…” to “Each party **shall** have an identifier…”. Also, inserted text that discusses the $EphemPubKey_i$ string and conversions to FCC and ECC schemes.
35. Section 5.9.1.1: Appended to Section 5.9.1, since there was no Section 5.9.1.2. Text was added to clarify the use of an ephemeral public key in the *MacData*.
36. Section 5.9.3: Modified text to approve the use of KMAC as a MAC algorithm. Removed the domain parameter-size sets, referring to Section 5.5.1 for the domain parameter information. Provided text specifying that the MacKey length needs to be at least the supported security strength of the domain parameters and the Mac tag length needs to be at least 64 bits. Also, added text and a table that identifies the approved MAC algorithms, *MacOutputLen*s and the security strengths that they can support.
37. Section 6.1.1: Modified the first assumption to refer to Section 5.5.1 for the domain parameter information. Now refer to Section 5.9.3 for the minimum *MacKey* and Mac tag lengths.
38. Section 6.1.1.1-6.1.1.4: Clarified error handling.
39. Section 6.1.2: Modified the first assumption to refer to Section 5.5.1 for the domain parameter information. Now refer to Section 5.9.3 for the minimum *MacKey* and Mac tag lengths.
40. Section 6.1.2.1-6.1.2.2: Clarified error handling.
41. Section 6.2.1: Modified the first assumption to refer to Section 5.5.1 for the domain parameter information. Now refer to Section 5.9.3 for the minimum *MacKey* and Mac tag lengths.
42. Section 6.2.1.1-6.2.1.4: Clarified error handling.
43. Section 6.2.2: Modified the first assumption to refer to Section 5.5.1 for the domain parameter information. Now refer to Section 5.9.3 for the minimum *MacKey* and Mac tag lengths.

44. Section 6.2.2.1-6.2.2.2: Clarified error handling.
45. Section 6.3: Modified the first assumption to refer to Section 5.5.1 for the domain parameter information. Now refer to Section 5.9.3 for the minimum *MacKey* and Mac tag lengths.
46. Section 6.31-6.3.2: Clarified error handling.
47. Section 7: Specified that the allowed methods for key wrapping are CCM, KW and KWP, and included subsections describing how to interface with them.

    Renamed *KeyWrappinKey* to *KWK*, *TransportedKeyingMaterial* to *KM* and *WrappedKeyingMaterial* to *WrappedKM*.

    Assumptions for DLC-based key-transport have been added.

    Added sections for using CCM (Sections 7.1 and 7.2), KW and KWP (Sections 7.1.3 and 7.1.4).
48. Section 10: Modified to refer to SP 800-56C for key-derivation methods.
49. Appendix A: Updated the FIPS and SP references.
50. Appendix B: Changed the title.
51. Appendix C.1: Changed the routine to specify the technique used in SP 800-56B; the same results should be obtained.
52. Appendix C.4: Added a bit string to integer conversion routine.
53. Appendix E: Inserted an appendix listing the **approved** safe-prime groups and a table providing various names for the NIST-recommended elliptic curves currently specified in FIPS 186-4. The curves will be moved to SP 800-186. The supported security strengths for the curves and the safe-prime groups is included in the tables.

Appendix E: Approved ECC Curves and FCC Safe-prime Groups

NIST will be providing lists of **approved** elliptic curves and FCC mod p groups in the FIPS 140 Implementation Guidance document, Section D.13 (IG D.13).

Elliptic Curves (EC) for Key Establishment: At this time, IG D.13 includes the following list of curves for use in the ECC DH and MQV key-establishment primitives, but does not include the associated targeted security strengths for which the use of each curve is appropriate.

Note: entries in the same row refer to the same EC under different names. Absence of equivalent entries is indicated by "-".

Referenced in:	**FIPS 186-4 SP 800-56A**	**TLS (RFC 4492) (SP 800-52)**	**IPsec w/ IKE v2 (RFC 5903)**	**Targeted Security Strengths that can be Supported**
Specified in:	SP 800-186[18]	SEC 2	RFC 5903	
	P-224	secp224r1	-	$s = 112$
	P-256	secp256r1	secp256r1	$112 \le s \le 128$
	P-384	secp384r1	secp384r1	$112 \le s \le 192$
	P-521	secp521r1	secp521r1	$112 \le s \le 256$
	K-233	sect233k1	-	$112 \le s \le 128$
	K-283	sect283k1	-	$112 \le s \le 128$
	K-409	sect409k1	-	$112 \le s \le 192$
	K-571	sect571k1	-	$112 \le s \le 256$
	B-233	sect233r1	-	$112 \le s \le 128$
	B-283	sect283r1	-	$112 \le s \le 128$
	B-409	sect409r1	-	$112 \le s \le 192$
	B-571	sect571r1	-	$112 \le s \le 256$

Finite Field Cryptography Groups for Key Establishment: The following safe-prime groups are defined in RFC 3526 and RFC 7919 for use with key-establishment schemes that employ either the FFC DH or FFC MQV primitives. IG D.13 currently lists the groups from RFC 3526, but does not list the groups from RFC 7919. The IG also does not identify the associated targeted security strengths for which the use of each group is appropriate.

The domain parameters for these groups have the form (p, $q = (p - 1)/2$, $g = 2$); the explicit values for p are provided in the RFCs.

[18] Specified in FIPS 186-4 until SP 800-186 is available.

IKE v2 (RFC 3526)	**Targeted Security Strengths that can be Supported**
MODP-2048 (ID=14)	$s = 112$
MODP-3072 (ID=15)	$112 \le s \le 128$
MODP-4096 (ID=16)	$112 \le s \le 152$*
MODP-6144 (ID=17)	$112 \le s \le 176$*
MODP-8192 (ID=18)	$112 \le s \le 200$*

TLS (RFC 7919)	**Targeted Security Strengths that can be Supported**
ffdhe2048 (ID = 256)	$s = 112$
ffdhe3072 (ID = 257)	$112 \le s \le 128$
ffdhe4096 (ID = 258)	$112 \le s \le 152$*
ffdhe6144 (ID = 259)	$112 \le s \le 176$*
ffdhe8192 (ID = 260)	$112 \le s \le 200$*

* The maximum security strength estimates were calculated using formula in Section 7.5 of the FIPS 140 IG and rounded to the nearest multiple of eight bits.