The Key Agreement Schemes Validation System (KASVS)

Updated: January 23, 2013
Previously Updated: September 1, 2011
Previously Updated: July 7, 2011

Sharon S. Keller

National Institute of Standards and Technology

Information Technology Laboratory

Computer Security Division

Update Log

1/23/13

Confirm that the format of the OtherInfo is out of scope of the CAVP testing. If Static or StaticUnified schemes is being tested, the CAVS testing does look for the $Nonce_U$ in the *PartyUInfo* subfield of *OtherInfo*.

8/30/11

Add Appendix B containing "shall" statements tested by CAVS.

7/7/11

1. Naming of files generated by the CAVS tool will contain KDFConcat or KDFASN1.
2. If Static scheme is being tested, the OtherInfo field used in the Key Derivation Function should contain Party U's nonce. Added a check to assure this nonce is in the OtherInfo field.
3. Added chart defining the variable names and definitions used in both the Function and Validity tests
4. Decided prerequisite information will not be included in the validation system document. It can be found on the webpage and in the FAQ.

3/2/11

1. It has been determined that the assurances are out of scope of the CAVP. Therefore the requirement to indicate the assurances has been removed. To assist in the validation testing and, possibly as a tool to determine the assurances, algorithmic functions used by the SP800-56A (but not defined in this special publication) are now required.

09/13/2010

1. 6.1 #3 x. and d. xi. Removed TDES from CCM line. CCM is for use with 128-bit block ciphers.

3/11/09

(Revised parts are underlined.)

2. Added specifications for testing the processing specified in SP800-56A through the calculation of the shared secret value (ZZ). This testing is provided for IUTs who have implemented SP800-56A but are using KDFs approved in IG D.2 **Acceptable Key Establishment Protocols** and not contained in SP800-56A.

3. Added an assurance to Section 5.6.2.3 that addresses the IG D.3 **Assurance of the Validity of a Public Key for Key Establishment** which states that an IUT can claim that the FFC or ECC Ephem scheme validation assurance is not required.

4. Added the requirement to enter the following configuration information:

4.1. The KDFs implemented:

4.1.1. Concatenation

4.1.2. ASN.1

4.2. The Nonce types used in key confirmation:

4.2.1. Random Nonce

4.2.2. Time Stamp

4.2.3. Monotonically increasing sequence number

4.2.4. Combination of 2 and 3

4.3. If Static scheme is supported, the Nonce types used in this scheme:

4.3.1. Random Nonce

4.3.2. Time Stamp

4.3.3. Monotonically increasing sequence number

4.3.4. Combination of 2 and 3

12/24/08

(Revised parts are underlined.)

1. Section 2 Scope, Paragraph about the prerequisites:

    a. The KASVS validation process also requires prerequisite testing of the underlying algorithms used in the implementation. They include:

        1. The underlying DSA and/or ECDSA algorithm's <u>domain parameter and/or key pair functions if the assurances selected indicate that this function should be in the implementation. Please refer to Table 1 in this document to determine what, if anything, needs to be tested as a prerequisite,</u>

        3. The supported MAC algorithms (CCM, CMAC, and/or HMAC) <u>if Key Confirmation is supported</u>, and

TABLE OF CONTENTS

1 Introduction

This document, *The Key Agreement Scheme (KAS) Validation System (KASVS)*, specifies the procedures involved in validating implementations of the Key Agreement Schemes. The testing encompasses IUTs that implement the key agreement schemes, as specified in SP 800-56A, *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography* [1], without key confirmation and with key confirmation. The KASVS is designed to perform automated testing on Implementations Under Test (IUTs).

In addition to testing the complete special publication, KASVS also provides testing of implementations of SP800-56A that implement a key derivation function NOT specified in SP800-56A (See IG 7.1). For this situation, the CAVP provides component testing of the special publication referred to as the testing of "all of SP800-56A except the key derivation function". This testing is performed in situations where an implementation of SP800-56A is using a KDF approved for use in Implementation Guidance D.2 **Acceptable Key Establishment Protocols** that is not specified in SP800-56A.

This document defines the purpose, the design philosophy, and the high-level description of the validation process for each key agreement scheme, either alone or accompanied with key confirmation and for the testing of the DLC primitive components. It includes specifications for the two categories of tests that make up the KASVS, i.e., the Function test and the Validity test. The requirements and administrative procedures to be followed by those seeking formal validation of an implementation of SP800-56A are presented. The requirements described include a specification of the data communicated between the IUT and the KASVS, the details of the tests that the IUT must pass for formal validation, and general instruction for interfacing with the KASVS.

A set of KAS test vectors is available on the http://csrc.nist.gov/cryptval/ website for testing purposes.

2 Scope

This document specifies the tests required to validate implementations of SP 800-56A for conformance to the key agreement schemes, either alone or accompanied with key confirmation, as specified in [1], and it specifies the tests required to validate implementations of "all of SP800-56A except the key derivation function". When applied to an Implementation Under Test (IUT), the KASVS provides testing to determine the correctness of the implementation of the key agreement scheme specifications and, if applicable, the key confirmation specifications. Determining the correctness of specifications in the IUT involves both the testing of the requirements identified by "shall" statements that are addressable at the algorithm level and requirements identified by the specifications in the standard.

As detailed in the Recommendation, Discrete Logarithm Cryptography (DLC) includes Finite Field Cryptography (FFC) and Elliptic Curve Cryptography (ECC). A separate validation test suite has been designed for each of these types of cryptography. These validation test suites contain validation testing for each key agreement scheme. The

validation testing verifies that an IUT has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the DLC primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the validation test suite also verifies that an IUT has implemented the components of key confirmation as specified in the Recommendation. This includes the parsing of the DKM, the generation of MacData and the calculation of MacTag. The requirements addressable at the algorithm level, and indicated by "shall" statements, that are tested by the validation test suite are listed in Appendix B.

If the IUT implements a KDF not specified in SP800-56A, the component testing of "all of SP800-56A except the key derivation function" is tested.

The KASVS validation process requires the definition of supporting cryptographic functions included within the implementation that are used by the SP800-56A but not defined in the special publication. These functions provide information to the KASVS to determine what validation testing is required. In addition, these functions may be used to assist in obtaining the assurances (but this is out of scope of the CAVP).

Note that the validation testing without key confirmation asks for a MAC algorithm to perform the testing. It is not a prerequisite to getting the IUT validated. Likewise, if "all of SP800-56A except the key derivation function" is being tested, the testing requires the shared secret value ZZ to be hashed. The hash function is only used to perform the test and is not a prerequisite to getting the IUT validated.

3 Conformance

The successful completion of the tests contained within the KASVS is required to claim conformance to SP800-56A. Testing for the cryptographic module in which a key agreement scheme(s) is implemented is defined in FIPS PUB 140-2, *Security Requirements for Cryptographic Modules*.[2]

4 Definitions and Abbreviations

4.1 Definitions

DEFINITION	**MEANING**
Assurance of identifier	Confidence that identifying information (such as a name) is correctly associated with an entity
Assurance of possession of a private key	Confidence that an entity possesses a private key associated with a public key.

Assurance of validity	Confidence that either a key or a set of domain parameters is arithmetically correct
CMT laboratory	Cryptographic Module Testing laboratory that operates the KASVS
Key agreement	A key establishment procedure where the resultant secret keying material is a function of information contributed by two participants, so that no party can predetermine the value of the secret keying material independently from the contributions of the other parties.
Key confirmation	A procedure to provide assurance to one party (the key confirmation recipient) that another party (the key confirmation provider) actually possesses the correct secret keying material and/or shared secret.

4.2 Abbreviations

ABBREVIATION	**MEANING**
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CCMVS	CCM Validation System
CMACVS	CMAC Validation System
DKM	Derived Keying Material
DLC	Discrete Logarithm Cryptography
DSA	Digital Signature Algorithm
DSAVS	Digital Signature Algorithm Validation System
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDSAVS	ECDSA Validation System
FIPS	Federal Information Processing Standard
HMAC	Keyed-Hash Message Authentication Code
HMACVS	HMAC Validation System
I.G. D.2	**Acceptable Key Establishment Protocols** identifies additional symmetric and asymmetric key establishment schemes allowed in a FIPS Approved mode of operation, in addition to those provided in SP 800-56A. As stated in this IG's resolution, in many cases, the KDF used to generate the keying material from the shared secret is

	not one of the KDFs specified in SP800-56A.
I.G. D.3	**Assurance of the Validity of a Public Key for Key Establishment**
IUT	Implementation Under Test
KAS	Key Agreement Scheme
KC	Key Confirmation
KDF	Key Derivation Function
KES	Key Establishment Scheme
MAC	Message Authentication Code
SHA	Secure Hash Algorithm
SHAVS	SHA Validation System
Z	A shared secret that is used to derive secret keying material using a key derivation function; a DLC primitive – either Diffie-Hellman or MQV.

5 Design Philosophy of Key Agreement Schemes Validation System

The KASVS is designed to test conformance to the key agreement and key confirmation specifications rather than provide a measure of a product's security. The validation tests are designed to assist in the detection of accidental implementation errors, and are not designed to detect intentional attempts to misrepresent conformance. Thus, validation should not be interpreted as an evaluation or endorsement of overall product security.

The KASVS has the following design philosophy:

1. The KASVS is designed to allow the testing of an IUT at locations remote to the KASVS. The KASVS and the IUT communicate data via *REQUEST* and *RESPONSE* files. The KASVS also generates *SAMPLE* files to provide the IUT with an example of the format required by the *RESPONSE* file.

2. The testing performed within the KASVS utilizes statistical sampling (i.e., only a small number of the possible cases are tested); hence, the successful validation of a device does not imply 100% conformance with the Recommendation.

6 Key Agreement Scheme Validation System (KASVS) Test

The KASVS tests the implementation of the key agreement and the key confirmation processes for its conformance to SP800-56A.

When applied to an IUT, the KASVS provides testing to determine the correctness of the implementation of the key agreement scheme specifications. As detailed in the Recommendation, Discrete Logarithm Cryptography (DLC) includes Finite Field Cryptography (FFC) and Elliptic Curve Cryptography (ECC). A separate validation test suite has been designed for each of these types of cryptography. Within each test suite, validation testing has been designed for each key agreement scheme. The validation test suite for each key agreement scheme verifies that an IUT has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the DLC primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the validation test suite also verifies that the components of key confirmation as specified in the Recommendation have been implemented correctly. This includes the parsing of the DKM, the generation of MacData and the calculation of MacTag.

Supporting cryptographic functions included within the implementation that are used by the SP800-56A but not defined in the special publication must be defined. These functions provide information to the KASVS to determine what validation testing is required. In addition, these functions may be used to assist in obtaining the assurances (but the assurances are out of scope of the CAVP). These functions include Domain Parameter Generation, Domain Parameter Validation, Key Pair Generation, Full Public Key Validation, Partial Public Key Validation (for ECC only) and Key Regeneration.

6.1 Configuration Information

To initiate the validation process of the KASVS, a vendor submits an application to an accredited laboratory requesting the validation of its implementation of the complete key agreement scheme with or without key confirmation, or only the processing up to and including the DLC primitive component. The vendor's implementation is referred to as the IUT. The request for validation includes background information describing the IUT, along with information needed by the KASVS to perform the specific tests. More specifically, the request for validation includes:

1. Cryptographic algorithm implementation information

    a. Vendor Name;

    b. Implementation Name;

    c. Implementation Version;

    d. Indication if implementation is software, firmware, or hardware;

e. Processor and Operating System with which the IUT was tested if the IUT is implemented in software or firmware;

f. Brief description of the IUT or the product/product family in which the IUT is implemented by the vendor (2-3 sentences); and

2. Configuration information for the KASVS tests.

   a. The underlying cryptographic schemes supported by the IUT, i.e., FFC and/or ECC. The FFC schemes are based on ANS X9.42 and the ECC schemes are based on ANS X9.63.

   b. For each underlying algorithm, a list of supporting cryptographic functions included within the IUT that are supported by SP800-56A but not defined in the special publication. Based on the functions defined as being supported by an IUT, the scope of the validation testing necessary to thoroughly test the implementation is determined.

3. If FFC is implemented, the following configuration information is required:

   a. Supported key agreement scheme(s):

      - dhHybrid1, MQV2, dhEphem, dhHybridOneFlow, MQV1, dhOneFlow, dhStatic

   b. Supported roles for key agreement:

      - Initiator, Responder

The following configuration information is required if the KDF implemented is specified in SP800-56A and key confirmation is supported:

   c. If key confirmation is supported, supported roles for key confirmation:

      - Provider, Recipient

   d. If key confirmation is supported, types of key confirmation:

      - Unilateral, Bilateral

   e. The KDFs implemented:

      - Concatenation
      - ASN.1

   f. The Nonce types used in key confirmation:

- Random Nonce
- Time Stamp
- Monotonically increasing sequence number
- Combination of 2 and 3

g. If Static scheme is supported, the Nonce types used in this scheme:

- Random Nonce
- Time Stamp
- Monotonically increasing sequence number
- Combination of 2 and 3

h. Parameter size set(s) supported:

- FA
- FB
- FC

(Refer to SP800-56A, Section 5.5.1.1, Table 1, FFC Parameter Size Sets for more information.)

i. SHA algorithm(s) supported for use in the key derivation function testing. For the testing of only the DLC primitive (ZZ Only), the SHA algorithm is requested for testing purposes only – to hash the ZZ value before outputting it. In this case, it is not required as a prerequisite to the SP800-56A implementation.

j. If key confirmation is supported, indicate all MACs supported by the IUT, along with the associated information. If key confirmation is not supported, indicate one MAC supported by the IUT, along with the associated information. Note in this case, the MAC is not required as a prerequisite to the SP800-56A implementation – it is only used in the testing process. The MACs to choose from are listed below:

- A NIST-approved MAC supported by the IUT:
  - CCM:
    - Algorithm: AES

- Key Size: 128, 192, 256
- Nonce Length in bytes: 7, 8, 9, 10, 11, 12, 13
- Tag Length in bytes:
    - For FA: 10, 12, 14, 16
    - For FB: 14, 16
    - For FC: 16

- CMAC:
    - Algorithm and key size: AES128, AES192, AES256
    - Tag Length in bytes:
        - For FA: 10 <= Tag Length <= 16
        - For FB: 14 <= Tag Length <=16
        - For FC: Tag Length = 16
- HMAC:
    - For FA:
        - SHA Algorithm supported: SHA1, SHA224, SHA256, SHA384, SHA512
        - HMAC Key Size in bytes: >= 10 bytes
        - Tag Length in bytes: >= 10 bytes
    - For FB:
        - SHA Algorithm supported: SHA224, SHA256, SHA384, SHA512
        - HMAC Key Size in bytes: >= 14 bytes
        - Tag Length in bytes: >= 14 bytes
    - For FC:

- SHA Algorithm supported: SHA256, SHA384, SHA512
- HMAC Key Size in bytes: >= 16 bytes
- Tag Length in bytes: >= 16 bytes

4. If ECC is implemented, the following configuration information is required:

    a. Supported key agreement scheme(s):

        - (Cofactor) Full Unified Model, Full MQV, (Cofactor) Ephemeral Unified Model, (Cofactor) One-Pass Unified Model, One-Pass MQV, (Cofactor) One-Pass Diffie-Hellman, Cofactor Static Unified Model

    b. Supported roles:

        - Initiator, Responder

The following configuration information is required if the KDF implemented is specified in SP800-56A and key confirmation is supported:

    c. If key confirmation is supported, supported roles for key confirmation:

        - Provider, Recipient

    d. If key confirmation is supported, supported types of key confirmation:

        - Unilateral, Bilateral

    e. The KDFs implemented:

        - Concatenation
        - ASN.1

    f. The Nonce types used in key confirmation:

        - Random Nonce
        - Time Stamp
        - Monotonically increasing sequence number
        - Combination of 2 and 3

    g. Static scheme is supported, the Nonce types used in this scheme:

- Random Nonce
- Time Stamp
- Monotonically increasing sequence number
- Combination of 2 and 3

h. Parameter set(s) supported:

- EA
- EB
- EC
- ED
- EE

(Refer to SP800-56A, Section 5.5.1.2, Table 2, ECC Parameter Size Sets for more information.)

i. Supported curve (indicate one per parameter set supported). Note, if an IUT supports both prime fields and polynomial fields, a parameter set from each field should be tested:

- For EA: P192, K163, B163
- For EB: P224, K233, B233
- For EC: P256, K283, B283
- For ED: P384, K409, B409
- For EE: P512, K571, B571

j. SHA algorithms supported for use in the key derivation function testing. For the testing of only the DLC primitive (ZZ Only), the SHA algorithm is requested for testing purposes only – to hash the ZZ value before outputting it. In this case, it is not required as a prerequisite to the SP800-56A implementation.

k. If key confirmation is supported, indicate all MACs supported by the IUT, along with the associated information. If key confirmation is not supported, indicate one MAC supported by the IUT, along with the associated information. Note in this case, the MAC is not required as a prerequisite to the SP800-56A implementation – it is only used in the testing process. The MACs to choose from are listed below:

- A NIST-approved MAC supported by the IUT:
  - CCM:
    - Algorithm: AES
    - Key Size: 128, 192, 256
    - Nonce Length in bytes: 7, 8, 9, 10, 11, 12, 13
    - Tag Length in bytes:
      - For EA: 10, 12, 14, 16
        - For EB: 14, 16
        - For EC: 16
  - CMAC (Only for use with EA, EB, EC):
    - Algorithm and key size: AES128, AES192, AES256
    - Tag Length in bytes:
      - For EA: 10 <= Tag Length <= 16
      - For EB: 14 <= Tag Length <=16
      - For EC: Tag Length = 16
  - HMAC:
    - For EA:
      - SHA Algorithm supported: SHA1, SHA224, SHA256, SHA384, SHA512
      - HMAC Key Size in bytes: >= 10 bytes
      - Tag Length in bytes: >= 10 bytes
    - For EB:
      - SHA Algorithm supported: SHA224, SHA256, SHA384, SHA512
      - HMAC Key Size in bytes: >= 14 bytes

        - Tag Length in bytes: >= 14 bytes

- For EC:
    - SHA Algorithm supported: SHA256, SHA384, SHA512
    - HMAC Key Size in bytes: >= 16 bytes
    - Tag Length in bytes: >= 16 bytes
- For ED:
    - SHA Algorithm supported: SHA384, SHA512
    - HMAC Key Size in bytes: >= 24 bytes
    - Tag Length in bytes: >= 24 bytes
- For EE:
    - SHA Algorithm supported: SHA512
    - HMAC Key Size in bytes: >= 32 bytes
    - Tag Length in bytes: >= 32 bytes

6.2 The Function Test

6.2.1 Key Confirmation Not Supported

A separate file is generated for each supported key agreement scheme – KDF type - role combination. For example, if an IUT supports the key agreement scheme dhHybrid1, uses KDF Concatenation and the IUT supports both initiator and responder roles, two files will be generated:

KASFunctionTest_FFCHybrid1_KDFConcat_NOKC_init.req and
KASFunctionTest_FFCHybrid1_KDFConcat_NOKC_resp.req.

Within each request file, there is a section for each combination of parameter set and SHA algorithm supported, i.e., FA-SHA1, FA-SHA224, FB-SHA224, FC-SHA512. For each combination of parameter set and SHA algorithm, the Function Test provides 10 sets of data to the IUT. In addition to this, if FFC is used, one set of domain parameter values is included for use with these 10 sets of data. If ECC is used, the curve name is included in the file header. Depending on the scheme being tested, this set of data may include a static public key and/or an ephemeral public key, and a nonce. The nonce is used in constructing the value of the MacData. (See Section 5.2.3 of NIST SP800-56A.)

The IUT uses the domain parameter values or the NIST-approved curves to generate a public/private key pair. The IUT uses the appropriate public keys supplied by the KASVS and its own public/private key pair to calculate the shared secret value *Z*. The *Z* value is computed using the appropriate DLC primitive corresponding to the scheme being tested (Section 5.7 of NIST SP800-56A).

The IUT also calculates the derived keying material *DKM*. SP800-56A specifies two key derivation functions in Section 5.8 - the Concatenation Key Derivation Function (Approved Alternative 1) and the ASN.1 Key Derivation Function (Approved Alternative 2). The *DKM* is computed using the supported KDFs. These two functions differ only in the format of the Other Information *OtherInfo* (OI) field. In the KASVS, the IUT is required to supply the value of the OI field since the exact format of this field is outside the scope of the algorithm specifications – i.e. it is application specific. This allows the CAVS tool to test both key derivation functions in the same manner. Note, for this reason, the format of the OI field is outside the scope of the KASVS validation testing. Other fields needed in the computation of the key derivation function are the IUTid, supplied by the IUT and the CAVSid, supplied by the CAVS tool. If Static scheme is being tested, Party U (which may be the IUT or the CAVS depending on the roles being tested) must supply a nonce.

The IUT computes a *Tag* to determine if the SP800-56A implementation has been implemented correctly. The IUT specifies an approved MAC algorithm supported by their implementation, i.e., CCM, CMAC, or HMAC. The MAC key is obtained from the *DKM*. The MacData to be MACed shall be the string "Standard Test Message" concatenated with the 16-byte nonce found in the request file (Section 5.2.3 of NIST SP800-56A).

The values generated by the IUT are stored in the *RESPONSE* file in the format specified in the *SAMPLE* file. There shall be a *RESPONSE* file for every *SAMPLE* file.

If the IUT indicates that they support full or partial validation of their keys, (denoted in the supporting cryptographic functions), the KASVS will perform a validation of the IUT's public keys. The KASVS will check to see if Party U's nonce is in the OI field if Static scheme is being tested. The KASVS will also verify the correctness of the IUT's *Tag* by calculating the shared secret value using the appropriate DLC primitive and the IUT's public keys, computing the derived keying material, and computing the *Tag*. The KASVS compares the IUT's *Tag* value to the KASVS *Tag* value to see if they are the same. If they are, then it can be determined that the implemented key agreement scheme, the DLC primitive implementation, and the KDF implementation are implemented correctly according to the Recommendation. If the values do not match, the IUT has an error in it. During the validation of the IUT, if an error occurs, the intermediate values generated by the CAVS, such as *Z* and *DKM*, are stored in the log file. The laboratory uses this information to assist the vendor in debugging their IUT.

6.2.2 Key Confirmation Supported

A separate file is generated for each supported combination of the key agreement scheme,

key agreement role, key confirmation role, key confirmation type and KDF type. For example, if an IUT supports FFC cryptography, the dhStatic key agreement scheme, both key agreement roles (initiator and responder), both key confirmation roles (provider and recipient), both key confirmation types (unilateral and bilateral), and both KDF types then 16 files will be generated:

KASFunctionTest_FFCStatic_KDFConcat_KC_init_prov_ulat.req
KASFunctionTest_FFCStatic_KDFConcat_KC_init_rcpt_ulat.req
KASFunctionTest_FFCStatic_KDFConcat_KC_init_prov_blat.req
KASFunctionTest_FFCStatic_KDFConcat_KC_init_rcpt_blat.req
KASFunctionTest_FFCStatic_KDFConcat_KC_resp_prov_ulat.req
KASFunctionTest_FFCStatic_KDFConcat_KC_resp_rcpt_ulat.req
KASFunctionTest_FFCStatic_KDFConcat_KC_resp_prov_blat.req
KASFunctionTest_FFCStatic_KDFConcat_KC_resp_rcpt_blat.req

KASFunctionTest_FFCStatic_KDFASN1_KC_init_prov_ulat.req
KASFunctionTest_FFCStatic_KDFASN1_KC_init_rcpt_ulat.req
KASFunctionTest_FFCStatic_KDFASN1_KC_init_prov_blat.req
KASFunctionTest_FFCStatic_KDFASN1_KC_init_rcpt_blat.req
KASFunctionTest_FFCStatic_KDFASN1_KC_resp_prov_ulat.req
KASFunctionTest_FFCStatic_KDFASN1_KC_resp_rcpt_ulat.req
KASFunctionTest_FFCStatic_KDFASN1_KC_resp_prov_blat.req
KASFunctionTest_FFCStatic_KDFASN1_KC_resp_rcpt_blat.req.

Within each *REQUEST* file, there is a section for each combination of parameter set and SHA algorithm supported, i.e., FA-SHA1, FA-SHA224, FB-SHA224, FC-SHA512. Within each combination of parameter set and SHA algorithm, the Function Test provides a section for each supported combination of MAC algorithm and key size, i.e., CCM AES128, CCM AES256. In addition to this, if FFC is used, one set of domain parameter values is included for use with these sets of data. If ECC is used, the curve name is included in the file header. In each MAC algorithm-key size section, the Function Test provides 10 sets of data to the IUT. Depending on the scheme being tested, this set of data may include a static public key and/or an ephemeral public key.

The IUT uses the domain parameter values or the NIST-approved curves to generate a public/private key pair. The IUT uses the appropriate public keys supplied by the KASVS and its own public/private key pair to calculate the shared secret value *Z*. The *Z* value is computed using the appropriate DLC primitive corresponding to the scheme being tested (Section 5.7 of NIST SP800-56A).

The IUT also calculates the derived keying material *DKM*. SP800-56A specifies two key derivation functions in Section 5.8 - the Concatenation Key Derivation Function (Approved Alternative 1) and the ASN.1 Key Derivation Function (Approved Alternative 2). The *DKM* is computed using the supported KDFs. These two functions differ only in the format of the Other Information *OtherInfo* (OI) field. In the KASVS, the IUT is required to supply the value of the OI field since the exact format of this field is outside the scope of the algorithm specifications – i.e. it is application specific. This allows the

CAVS tool to test both key derivation functions in the same manner. Note, for this reason, the format of the OI field is outside the scope of the KASVS validation testing. Other fields needed in the computation of the key derivation function are the IUTid, supplied by the IUT and the CAVSid, supplied by the CAVS tool. If Static scheme is being tested, Party U (which may be the IUT or the CAVS depending on the roles being tested) must supply a nonce.

The IUT computes *Tags* for each implemented approved MAC algorithm supported by their implementation. These include CCM, CMAC, and/or HMAC. For each supported MAC algorithm, a MAC key will be obtained from the DKM. Depending on the key confirmation role (provider or recipient) and the key confirmation type (unilateral or bilateral), MacData will be computed as specified in Section 8 of NIST SP800-56A.

The values generated by the IUT are stored in the *RESPONSE* file in the format specified in the *SAMPLE* file. There shall be a *RESPONSE* file for every *SAMPLE* file.

If the IUT indicates that they support full or partial validation of their keys, (denoted in the supporting cryptographic functions), the KASVS will perform a validation of the IUT's public keys. The KASVS will check to see if Party U's nonce is in the OI field if Static scheme is being tested. The KASVS will also verify the correctness of the IUT's *Tag* by calculating the shared secret value using the appropriate DLC primitive and the IUT's public keys, computing the derived keying material, computing the MacData value, and computing the *Tag*. The KASVS compares the IUT's *Tag* value to the KASVS *Tag* value to see if they are the same. If they are, then it can be determined that the implemented key agreement scheme, the DLC primitive implementation, and the KDF implementation are implemented correctly according to the Recommendation. If the values do not match, the IUT has an error in it. During the validation of the IUT, if an error occurs, the intermediate values generated by the CAVS, such as *Z*, MacData, and *DKM*, are stored in the log file. The laboratory uses this information to assist the vendor in debugging their IUT.

6.2.3 Testing of the 800-56A Processing through Shared Secret Computation (ZZ) for IUTs that do not use a KDF specified in 800-56A

A separate file is generated for each supported key agreement scheme - role combination. For example, if an IUT supports the key agreement scheme dhHybrid1, and the IUT supports both initiator and responder roles, two files will be generated:

KASFunctionTest_FFCHybrid1_NOKC_ZZOnly_init.req and
KASFunctionTest_FFCHybrid1_NOKC_ ZZOnly_resp.req.

Within each request file, there is a section for each combination of parameter set and SHA algorithm supported, i.e., FA-SHA1, FA-SHA224, FB-SHA224, FC-SHA512. For each combination of parameter set and SHA algorithm, the Function Test provides 10 sets of data to the IUT. In addition to this, if FFC is used, one set of domain parameter values is included for use with these 10 sets of data. If ECC is used, the curve name is

included in the file header. Depending on the scheme being tested, this set of data may include a static public key and/or an ephemeral public key.

The IUT uses the domain parameter values or the NIST-approved curves to generate a public/private key pair. The IUT uses the appropriate public keys supplied by the KASVS and its own public/private key pair to calculate the shared secret value *Z*. The *Z* value is computed using the appropriate DLC primitive corresponding to the scheme being tested (Section 5.7 of NIST SP800-56A). Because IUTs shall not output the *Z* value in the clear, the *Z* value is hashed. Note that this hash implementation is for testing purposes only. It is not a prerequisite for this testing.

The values generated by the IUT are stored in the *RESPONSE* file in the format specified in the *SAMPLE* file. There shall be a *RESPONSE* file for every *SAMPLE* file.

If the IUT indicates that they support full or partial validation of their keys, (denoted in the supporting cryptographic functions), the KASVS will perform a validation of the IUT's public keys. The KASVS will also verify the correctness of the IUT's secret value using the appropriate DLC primitive and the IUT's public keys. The KASVS compares the IUT's *hashed Z* value to the KASVS *hashed Z* value to see if they are the same. If they are, then it can be determined that the implemented key agreement scheme and the DLC primitive implementation are implemented correctly according to the Recommendation. If the values do not match, the IUT has an error in it. During the validation of the IUT, if an error occurs, the intermediate values of *Z* generated by the CAVS are stored in the log file. The laboratory uses this information to assist the vendor in debugging their IUT.

6.2.4 Definition of Variables used in CAVS files for Function test

The Function test uses many variables as denoted in the request and sample files. Below is a table for the FFC Function test variables and the ECC Function test variables.

6.2.4.1 FFC Function Test Variables

Table 1: FFC Function Test Variables for all schemes for with key confirmation (KC) and without key confirmation (NOKC)

P	Domain parameter for DSA
Q	Domain parameter for DSA
G	Domain parameter for DSA
COUNT	the number of the set of values to be tested by IUT. Used to identify each set of values.
YstatCAVS	CAVS DSA static public key

YephemCAVS	CAVS DSA ephemeral public key
Nonce	ONLY USED BY NOKC. the 16-byte nonce that is to be concatenated to the message "Standard Test Message" to make the value of MacData. This MacData value is used for purposes of testing when no key confirmation capability exists. See Section 5.2.3 Implementation Validation Message.
CCMNonce	nonce used by the CCM function, if CCM is used to generate the Tag.
YstatIUT	IUT DSA static public key
YephemIUT	IUT DSA ephemeral public key
OILen	Length of the OtherInfo field. See Section 5.8 Key Derivation Functions for Key Agreement Schemes.
OI	OtherInfo is the bit string defined for the Concatenation and ASN.1 Key Derivation Functions. It must include at least the initiator's id, the responder's id, and, if Static Scheme is being tested, the initiator's nonce. See Section 5.8 Key Derivation Functions for Key Agreement Schemes, OtherInfo definition.
IUTidLen	the length of the IUT's id
IUTid	The IUT's id
DKM	Derived Keying Materal generated by running one of the Key Derivation Functions specified in SP800-56A. See Section 5.8.
Tag	The tag (or MAC) generated by using the DKM to MAC the Message with the specified method (CCM, CMAC, HMAC).
NonceEphemCAVS	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.

NonceDKMLen	ONLY USED BY STATIC SCHEME. the length of the nonce supplied by the initiator to be used in the OI field in the PartyUInfo field.
NonceDKM	ONLY USED BY STATIC SCHEME. The nonce supplied by the initiator to be used in the OI field in the PartyUInfo field. See Section 6.3.1 dhStatic, Action 4. In Table 18, Row "Ephemeral Data" it is referred to as Nonceu. In the same table, row "Derive Secret Keying Material" it states "Compute kdf(Z, OtherInput) using Nonceu". If the IUT is being tested for the Initiator role, the IUT supplies this value. If the IUT is being tested for the Responder role, the CAVS supplies this value.
NonceEphemIUTLen	length of NonceEphemIUT value
NonceEphemIUT	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.
Message	ONLY USED BY NOKC The complete message to be MACed which includes the message "Standard Test Message" concatenated with the Nonce above. See Section 5.2.3 Implementation Validation Message.
MacData	ONLY USED BY KC. The message to be MAced. See Section 8 Key Confirmation for the specific format of this field based on the scheme, key confirmation role, the key agreement role, and whether unilateral or bilateral key confirmation is being tested.

6.2.4.2 ECC Function Test Variables

Table 2: ECC Function Test Variables for all schemes for with key confirmation (KC) and without key confirmation (NOKC)	
COUNT	the number of the set of values to be tested by IUT. Used to identify each set of values.
QsCAVSx	CAVS ECDSA static public key x

	coordinate
QsCAVSy	CAVS ECDSA static public key y coordinate
QeCAVSx	CAVS ECDSA ephemeral public key x coordinate
QeCAVSy	CAVS ECDSA ephemeral public key y coordinate
Nonce	the 16-byte nonce that is to be concatenated to the message "Standard Test Message" to make the value of MacData. This MacData value is used for purposes of testing when no key confirmation capability exists. See Section 5.2.3 Implementation Validation Message.
CCMNonce	nonce used by the CCM function, if CCM is used to generate the Tag.
QsIUTx	IUT ECDSA static public key x coordinate
QsIUTy	IUT ECDSA static public key y coordinate
QeIUTx	IUT ECDSA ephemeral public key x coordinate
QeIUTy	IUT ECDSA ephemeral public key y coordinate
OILen	Length of the OtherInfo field. See Section 5.8 Key Derivation Functions for Key Agreement Schemes.
OI	OtherInfo is the bit string defined for the Concatenation and ASN.1 Key Derivation Functions. It must include at least the initiator's id, the responder's id, and, if Static Scheme is being tested, the initiator's nonce. See Section 5.8 Key Derivation Functions for Key Agreement Schemes, OtherInfo definition.
IUTidLen	the length of the IUT's id
IUTid	The IUT's id
DKM	Derived Keying Material generated by running one of the Key Derivation Functions specified in SP800-56A. See Section 5.8.
Tag	The tag (or MAC) generated by using the DKM to MAC the Message with the specified method (CCM, CMAC, HMAC).
NonceEphemCAVS	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.
NonceDKMLen	the length of the nonce supplied by the

	initiator to be used in the OI field in the PartyUInfo field.
NonceDKM	the nonce supplied by the initiator to be used in the OI field in the PartyUInfo field. See Section 6.3.1 dhStatic, Action 4. In Table 18, Row "Ephemeral Data" it is referred to as Nonceu. In the same table, row "Derive Secret Keying Material"it states "Compute kdf(Z, OtherInput) using Nonceu". If the IUT is being tested for the Initiator role, the IUT supplies this value. If the IUT is being tested for the Responder role, the CAVS supplies this value.
NonceEphemIUTLen	length of NonceEphemIUT value
NonceEphemIUT	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.
Message	ONLY USED BY NOKC The complete message to be MACed which includes the message "Standard Test Message" concatenated with the Nonce above. See Section 5.2.3 Implementation Validation Message.
MacData	ONLY USED BY KC. The message to be MAced. See Section 8 Key Confirmation for the specific format of this field based on the scheme, key confirmation role, the key agreement role, and whether unilateral or bilateral key confirmation is being tested.

6.3 The Validity Test

The second test in the NIST SP800-56A suite of validation tests is the Validity test. Its purpose is to test the ability of the IUT to recognize valid and invalid results received from the CAVS tool generated by the key agreement process with or without key confirmation. Incorrect values are generated by the CAVS tool by interjecting errors in different fields. The fields in which errors are introduced include *Z*, *DKM*, *OI*, *MacData*, *Tag*, CAVS' static public key, IUT's static public key, CAVS ephemeral pubic key and the IUT's static private key. Errors introduced in the keys test if the IUT has implemented the pubic key validation function properly. Note that this is only performed if the cryptographic functions supported by the IUT support this capability.

A version of this test is also included for IUTs that implement all of SP800-56A except a KDF specified in SP800-56A.

6.3.1 Key Confirmation Not Supported

A separate file is generated for each supported key agreement scheme – KDF type - role combination. For example, if an IUT supports the ECC key agreement scheme dhFullUnified, uses KDF Concatenation and the IUT supports both initiator and responder roles, two files will be generated:

KASValidityTest_ECCFullUnif_KDFConcat_NOKC_init.req and
KASValidityTest_ECCFullUnif_KDFConcat_NOKC_resp.req.

Within each request file, there is a section for each combination of parameter set and SHA algorithm supported (for example, FA-SHA1, FA-SHA224, FB-SHA224, FC-SHA512). For each combination of parameter set and SHA algorithm, the Validity Test provides information identifying the domain parameter values (for FFC) or elliptic curve (ECC) being used. For FFC implementations, the KASVS will generate 24 sets of data for the IUT. For ECC implementations, the KASVS will generate 30 sets of data for the IUT. Within these sets of data, the KASVS will modify some of the values to introduce errors. This will determine whether or not the IUT can detect these errors. In addition to verifying that the IUT can detect errors in the key agreement and key confirmation processing, this test will also provide assurance of the validity of the domain parameters as implemented by the IUT.

Depending on the key agreement scheme being tested, data supplied by the CAVS includes:

1 A header containing:

   a. Parameter Sets Supported

   b. CAVSid

   c. IUTid

   d. The parameters associated with each parameter set, including:

      i. Curve selected (if ECC)

      ii. SHA(s) supported

      iii. MAC algorithm(s) supported

      iv. If the MAC is CCM:

         1. Key sizes supported

         2. CCM Nonce length

         3. CCM Tag length

v. If the MAC is CMAC:

1. Key sizes supported

2. AES/TDES Tag length

vi. If the MAC is HMAC:

1. SHA(s) supported

2. Key sizes supported

3. Tag length

2 A set of data containing a subset of the following data depending on the scheme implemented:

a. CAVS values, including:

i. Static public key and/or

ii. Ephemeral public key (or nonce)

b. Nonce value

c. IUT values, including:

i. Static private key, and

ii. Static public key, and/or

iii. Ephemeral private key, and

iv. Ephemeral public key

d. If CCM is selected: the CCMNonce value

e. Other Information, OI

f. CAVS Tag

The IUT uses this information to validate the CAVS Tag value, returning a (P)ASS or (F)AIL. The IUT generates a response file containing the values above, plus the tag generated by the IUT (IUTTag) and the Result. The format for the *RESPONSE* file is specified in the *SAMPLE* file. There shall be a *RESPONSE* file for every *SAMPLE* file.

The KASVS verifies that the correct responses were returned by the IUT by comparing the results in the *RESPONSE* file with those in the *FAX* file. If the results match, CAVS records (P)ASS for this test; otherwise, CAVS records (F)AIL.

6.3.2 Key Confirmation Supported

A separate file is generated for each supported combination of the key agreement scheme, KDF type, key agreement role, key confirmation role and key confirmation type. For example, if an IUT supports the FFC key agreement scheme dhHybrid1, the Concatenation KDF, the key agreement role of initiator, both key confirmation roles (provider and recipient), and both key confirmation types (unilateral and bilateral), four files will be generated:

KASValidityTest_FFCHybrid1_KDFConcat_KC_init_prov_ulat.req
KASValidityTest_FFCHybrid1_KDFConcat_KC_init_prov_blat.req

KASValidityTest_FFCHybrid1_KDFConcat_KC_init_rcpt_ulat.req
KASValidityTest_FFCHybrid1_KDFConcat_KC_init_rcpt_blat.req

Within each *REQUEST* file, there is a section for each combination of parameter set and SHA algorithm supported, i.e., FA-SHA1, FA-SHA224, FB-SHA224, FC-SHA512. Within each combination of parameter set and SHA algorithm, the Validity Test provides a section for each supported combination of MAC algorithm and key size, i.e., CCM AES128, CCM AES256. In addition to this, if FFC is used, one set of domain parameter values is included for use with these sets of data. If ECC is used, the curve name is included in the file header.

In each MAC algorithm-key size section, the Validity Test generates 24 sets of data for FFC implementations and 30 sets of data for ECC implementations. Within these sets of data, the KASVS alters some of the values to introduce errors. This will determine whether or not the IUT can detect these errors. In addition to verifying that the IUT can detect errors in the key agreement and key confirmation processing, this test will also provide assurance of the validity of the domain parameters if implemented by the IUT.

Depending on the key agreement scheme being tested, data supplied by the CAVS includes:

3 A header containing:

a. Parameter Sets Supported

b. CAVSid

c. IUTid

d. Key Confirmation Types Supported

e. The parameters associated with each parameter set, including:

i. Curve selected (if ECC)

ii. SHA(s) supported

iii. MAC algorithm(s) supported

iv. If the MAC is CCM:

1. Key sizes supported

2. CCM Nonce length

3. CCM Tag length

v. If the MAC is CMAC:

1. Key sizes supported

2. AES/TDES Tag length

vi. If the MAC is HMAC:

1. SHA(s) supported

2. Key sizes supported

3. Tag length

4 A set of data containing a subset of the following data depending on the scheme implemented:

a. CAVS values, including:

i. Static public key and/or

ii. Ephemeral public key (or nonce)

b. IUT values, including:

i. Static private key, and

ii. Static public key, and/or

iii. Ephemeral private key, and

iv. Ephemeral public key

c. If CCM is selected: the CCMNonce value

d. Other Information, OI

e. CAVS Tag

The IUT uses this information to validate the CAVS Tag value returning a (P)ASS or (F)AIL. The IUT generates a response file containing the values above, plus the tag generated by the IUT (IUTTag) and the Result. The format for the *RESPONSE* file is specified in the *SAMPLE* file. There shall be a *RESPONSE* file for every *SAMPLE* file.

The KASVS compares the contents of the *RESPONSE* file with the contents of the *FAX* file. If the results match, CAVS records (P)ASS for this test; otherwise, CAVS records (F)AIL.

6.3.3 Testing of the 800-56A Processing through Shared Secret Computation (ZZ) for IUTs that do not use a KDF specified in 800-56A

A separate file is generated for each supported key agreement scheme - role combination. For example, if an IUT supports the ECC key agreement scheme dhFullUnified, and the IUT supports both initiator and responder roles, two files will be generated:

KASValidityTest_ECCFullUnif_NOKC_ZZOnly_init.req and
KASValidityTest_ECCFullUnif_NOKC_ ZZOnly_resp.req.

Within each request file, there is a section for each combination of parameter set and SHA algorithm supported (for example, FA-SHA1, FA-SHA224, FB-SHA224, FC-SHA512). For each combination of parameter set and SHA algorithm, the Validity Test provides information identifying the domain parameter values (for FFC) or elliptic curve (ECC) being used. For FFC implementations, the KASVS will generate 24 sets of data for the IUT. For ECC implementations, the KASVS will generate 30 sets of data for the IUT. Within these sets of data, the KASVS will modify some of the values to introduce errors. This will determine whether or not the IUT can detect these errors. In addition to verifying that the IUT can detect errors in the key agreement processing, this test will also provide assurance of the validity of the domain parameters as implemented by the IUT.

Depending on the key agreement scheme being tested, data supplied by the CAVS includes:

1 A header containing:

a. Parameter Sets Supported

b. The parameters associated with each parameter set, including:

i. Curve selected (if ECC)

ii. SHA(s) used for hashing Z (this is only for testing purposes. It is not a prerequisite for testing “All of 800-56A except KDF”.)

2 A set of data containing a subset of the following data depending on the scheme implemented:

a. CAVS values, including:

i. Static public key and/or

ii. Ephemeral public key (or nonce)

b. IUT values, including:

i. Static private key, and

ii. Static public key, and/or

iii. Ephemeral private key, and

iv. Ephemeral public key

c. Hash of the CAVS Z value (CAVSHashZZ)

The IUT uses this information to validate the CAVS Z value, returning a (P)ASS or (F)AIL. The IUT generates a response file containing the values above, plus the Z value generated by the IUT (IUTHashZZ) and the Result. The format for the *RESPONSE* file is specified in the *SAMPLE* file. There shall be a *RESPONSE* file for every *SAMPLE* file.

The KASVS verifies that the correct responses were returned by the IUT by comparing the results in the *RESPONSE* file with those in the *FAX* file. If the results match, CAVS records (P)ASS for this test; otherwise, CAVS records (F)AIL.

6.4.4 Definition of Variables used in CAVS files for Validity test

The Function test uses many variables as denoted in the request and sample files. Below is a table for the FFC Function test variables and the ECC Function test variables.

6.2.4.1 FFC Validity Test Variables

Table 3: FFC Validity Test Variables for all schemes for with key confirmation (KC) and without key confirmation (NOKC)	
P	Domain parameter for DSA (supplied by CAVS)
Q	Domain parameter for DSA (supplied by CAVS)
G	Domain parameter for DSA (supplied by CAVS)
COUNT	the number of the set of values to be tested by IUT. Used to identify each set of values. (supplied by CAVS)
YstatCAVS	CAVS DSA static public key (supplied by

	CAVS)(Not used by dhEphem)
YephemCAVS	CAVS DSA ephemeral public key (supplied by CAVS)
NonceEphemCAVS	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.
Nonce	ONLY USED BY NOKC. the 16-byte nonce that is to be concatenated to the message "Standard Test Message" to make the value of MacData. This MacData value is used for purposes of testing when no key confirmation capability exists. See Section 5.2.3 Implementation Validation Message. (supplied by CAVS)
NonceDKMCAVS	ONLY USED BY STATIC SCHEME. the nonce supplied by the initiator to be used in the OI field in the PartyUInfo field. See Section 6.3.1 dhStatic, Action 4. In Table 18, Row "Ephemeral Data" it is referred to as Nonceu. In the same table, row "Derive Secret Keying Material" it states "Compute kdf(Z, OtherInput) using Nonceu". If the IUT is being tested for the Responder role, this value is associated with the CAVS.
XstatIUT	IUT DSA static private key (supplied by CAVS)(Not used by dhEphem)
YstatIUT	IUT DSA static public key (supplied by CAVS)(Not used by dhEphem)
NonceEphemIUT	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.
XephemIUT	IUT DSA ephemeral private key (supplied by CAVS)
YephemIUT	IUT DSA ephemeral public key (supplied by CAVS)
NonceDKMIUT	ONLY USED BY STATIC SCHEME. the nonce supplied by the initiator to be used in the OI field in the PartyUInfo field. See Section 6.3.1 dhStatic, Action 4. In Table 18, Row "Ephemeral Data" it is referred to as Nonceu. In the same table, row "Derive Secret Keying Material" it states "Compute kdf(Z, OtherInput) using Nonceu". If the IUT is being tested for the Initiator role, this value is associated with the IUT.
CCMNonce	nonce used by the CCM function, if CCM is

	used to generate the Tag.(supplied by IUT)
OI	OtherInfo is the bit string defined for the Concatenation and ASN.1 Key Derivation Functions. It must include at least the initiator's id, the responder's id, and, if Static Scheme is being tested, the initiator's nonce. See Section 5.8 Key Derivation Functions for Key Agreement Schemes, OtherInfo definition. (supplied by IUT)
CAVSTag	The tag (or MAC) GENERATED BY THE CAVS by using the DKM to MAC the Message with the specified method (CCM, CMAC, HMAC).(supplied by CAVS)
IUTTag	The tag (or MAC) GENERATED BY THE IUT by using the DKM to MAC the Message with the specified method (CCM, CMAC, HMAC).(supplied by IUT)
Result	P (Pass) or F (Fail) indicating if the IUT agrees with the Tag generated by the CAVS.

6.2.4.2 ECC Validity Test Variables

Table 4: ECC Validity Test Variables for all schemes for with key confirmation (KC) and without key confirmation (NOKC)	
COUNT	the number of the set of values to be tested by IUT. Used to identify each set of values.
QsCAVSx	CAVS ECDSA static public key x coordinate
QsCAVSy	CAVS ECDSA static public key y coordinate
QeCAVSx	CAVS ECDSA ephemeral public key x coordinate
QeCAVSy	CAVS ECDSA ephemeral public key y coordinate
NonceEphemCAVS	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.
Nonce	the 16-byte nonce that is to be concatenated to the message "Standard Test Message" to make the value of MacData. This MacData value is used for purposes of testing when no key confirmation capability exists. See Section 5.2.3 Implementation Validation Message.
NonceDKMCAVS	the nonce supplied by the initiator to be used in the OI field in the PartyUInfo field. See Section 6.3.1 dhStatic, Action 4. In Table 18, Row "Ephemeral Data" it is referred to as Nonceu. In the same table, row "Derive Secret Keying Material" it states "Compute kdf(Z, OtherInput) using Nonceu". If the IUT is being tested for the Responder role, this value is associated with the CAVS.
dsIUT	IUT ECDSA static private key
QsIUTx	IUT ECDSA static public key x coordinate
QsIUTy	IUT ECDSA static public key y coordinate
NonceEphemIUT	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field.
deIUT	IUT ECDSA ephemeral private key
QeIUTx	IUT ECDSA ephemeral public key x coordinate
QeIUTy	IUT ECDSA ephemeral public key y coordinate
NonceDKMIUT	the nonce supplied by the initiator to be used in the OI field in the PartyUInfo field. See Section 6.3.1 dhStatic, Action 4. In Table 18, Row "Ephemeral Data" it is referred to as Nonceu. In the same table, row "Derive Secret Keying Material"it states "Compute kdf(Z, OtherInput) using Nonceu". If the IUT is being tested for the

	Initiator role, this valueis associated with the IUT.
CCMNonce	nonce used by the CCM function, if CCM is used to generate the Tag. (supplied by IUT)
OI	OtherInfo is the bit string defined for the Concatenation and ASN.1 Key Derivation Functions. It must include at least the initiator's id, the responder's id, and, if Static Scheme is being tested, the initiator's nonce. See Section 5.8 Key Derivation Functions for Key Agreement Schemes, OtherInfo definition. (supplied by IUT)
CAVSTag	The tag (or MAC) GENERATED BY THE CAVS by using the DKM to MAC the Message with the specified method (CCM, CMAC, HMAC).(supplied by CAVS)
IUTTag	The tag (or MAC) GENERATED BY THE IUT by using the DKM to MAC the Message with the specified method (CCM, CMAC, HMAC).(supplied by IUT)
Result	P (Pass) or F (Fail) indicating if the IUT agrees with the Tag generated by the CAVS.

Appendix A References

[1] *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, Special Publication 800-56A, National Institute of Standards and Technology, March 2006.

[2] *Security Requirements for Cryptographic Modules*, FIPS Publication 140-2, National Institute of Standards and Technology, May 2001.

Appendix B Requirements Identified By "Shall" Statements That Are Tested by the CAVP validation testing

The "shall" statements in all special publications indicate requirements that must be fulfilled to claim conformance to this Recommendation. The "shall" statements in the Special Publications address requirements at the algorithm, module, product level, and/or a higher level.

This section identifies the "shall" statements tested at the algorithm level when performing the KAS validation test suite.

5. Cryptographic Elements

All cryptographic elements used together **shall** be selected in accordance with the same parameter size set.

CAVS only tests allowable combinations. An IUT must implement at least one allowable combination of cryptographic elements to be testable.

5.1 Cryptographic Hash Functions

An Approved hash function **shall** be used when a hash function is required (for example, for the key derivation function or to compute a MAC when HMAC, as specified in FIPS 198, is used). FIPS 180-2 [2] specifies Approved hash functions.

Hash validation testing is a prerequisite to KAS validation.

The hash function **shall** be selected in accordance with the parameter lists in Tables 1 and 2 of Section 5.5.

CAVS only tests allowable combinations. An IUT must implement at least one allowable combination of cryptographic elements to be testable.

5.2 Message Authentication Code (MAC) Algorithm

An Approved MAC algorithm **shall** be used to compute a *MacTag*, for example, HMAC [5].

MAC validation testing is a prerequisite to KAS validation.

5.2.1 *MacTag* Computation

The *MacTag* computation **shall** be performed using an Approved MAC algorithm.

MAC validation testing is a prerequisite to KAS validation.

5.2.3 Implementation Validation Message

For purposes of validating an implementation of the schemes in this Recommendation during an implementation validation test (under the NIST Cryptographic Validation Program), the value of *MacData* **shall** be the string "Standard Test Message", followed by a 16-byte field for a nonce.

CAVS supplies a random nonce for each set of test values. The specified string concatenated with the random nonce is the value used as the MacData for testing implementations with no key confirmation.

5.3 Random Number Generation

Whenever this Recommendation requires the use of a randomly generated value (for example, for keys or nonces), the values **shall** be generated using an Approved random bit generator (RBG) providing an appropriate security strength.

Random number generator (RNG/DRBG) validation testing is a prerequisite to KAS validation.

5.4 Nonces

The security strength of the random bit generator and the entropy of the nonce **shall** be at least one half of the minimum required bit length of the subgroup order (as specified in Tables 1 and 2 of Section 5.5).

CAVS testing requires the nonce to be at least one half of the minimum required bit length of the subgroup order.

5.5 Domain Parameters

For this Recommendation, however, only one set of domain parameters **shall** be used during any key establishment transaction using a given run of a scheme (that is, the statickey domain parameters and the ephemeral-key domain parameters used in one scheme **shall** be the same).

CAVS only tests one set of domain parameters per key establishment transaction at one time.

5.5.1.1 FFC Domain Parameter Generation

FFC Domain parameters **shall** be generated using a method specified in FIPS 186-3 [3] based on a parameter size set selected from Table 1.

DSA validation testing is a prerequisite to KAS validation.

As shown in Table 1, there are three parameter size sets (named FA through FC) for FFC; all the parameters of a particular set **shall** be used together.

CAVS only tests allowable combinations. An IUT must implement at least one allowable combination of cryptographic elements to be testable.

5.5.1.2 ECC Domain Parameter Generation

As shown in Table 2, there are five parameter size sets (named EA, EB, EC, ED and EE) for ECC; all the members of a particular set **shall** be used together.

CAVS only tests allowable combinations. An IUT must implement at least one allowable combination of cryptographic elements to be testable.

The ECC domain parameters **shall** either be generated as specified in ANS X9.62 [13] or selected from the recommended elliptic curve domain parameters specified in FIPS 186-3 [3].

ECDSA validation testing is a prerequisite to KAS validation.

5.6.1.1 FFC Key Pair Generation

For the FFC schemes, each static and ephemeral private key and public key **shall** be generated using an Approved method and the selected valid domain parameters (*p, q, g*{, *SEED, pgenCounter*}) (see Appendix B of FIPS 186-3).

DSA Key Pair Generation is a prerequisite to KAS validation.

Each private key **shall** be unpredictable and **shall** be generated in the range [1, *q*-1] using an Approved random bit generator.

RNG and/or DRBG are prerequisites to KAS validation.
DSA is a prerequisite to KAS validation to assure private key is in range

[1,q-1].

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 and the selected domain parameters (*q, FR, a, b*{*, SEED*}*, G, n, h*) (see Appendix B of FIPS 186-3).

ECDSA Key Pair Generation and Key Pair Verification are prerequisites to KAS validation.

Each private key, *d*, **shall** be unpredictable and **shall** be generated in the range [1, *n*-1] using an Approved random bit generator.

RNG and/or DRBG are prerequisites to KAS validation.
ECDSA is a prerequisite to KAS validation to assure private key is in range [1, n-1].

5.6.2.1 Owner Assurances of Static Public Key Validity

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

1. Owner Full Validation - The owner performs a successful full public key validation (see Sections 5.6.2.4 and 5.6.2.5). For example, a key generation routine may perform full public key validation as part of its processing.
2. TTP Full Validation – The owner receives assurance that a trusted third party (trusted by the owner) has performed a successful full public key validation (see Sections 5.6.2.4 and 5.6.2.5).
3. Owner Generation – The owner has generated the public key from the private key (see Section 5.6.1).
4. TTP Generation – The owner has received assurance that a trusted third party (trusted by the owner) has generated the public/private key pair and has provided the key pair to the owner (see Section 5.6.1).

Out of scope of the algorithm testing. However, if the IUT has the supporting function key pair generation or full public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

5.6.2.2 Recipient Assurances of Static Public Key Validity

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

1. Recipient Full Validation - The recipient performs a successful full public key validation (see Sections 5.6.2.4 and 5.6.2.5).
2. TTP Full Validation – The recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful full public key validation (see Sections 5.6.2.4 and 5.6.2.5).
3. TTP Generation – The recipient receives assurance that a trusted third party (trusted by the recipient) has generated the public/private key pair in accordance with Section 5.6.1and has provided the key pair to the owner.

Out of scope of the algorithm testing. However, if the IUT has the supporting function full public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

5.6.2.3 Recipient Assurances of Ephemeral Public Key Validity

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

1. Recipient Full Validation - The recipient performs a successful full public key validation (see Sections 5.6.2.4 and 5.6.2.5).
2. TTP Full Validation – The recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful full public key validation (see Sections 5.6.2.4 and 5.6.2.5). For example, a trusted processor may only forward an ephemeral public key to the recipient if the public key passes a full public key validation.
3. Recipient ECC Partial Validation - If using an ECC method (only), the recipient performs a successful partial public key validation (see Section 5.6.2.6).
4. TTP ECC Partial Validation – If using an ECC method (only), the recipient receives assurance that a trusted third party (trusted by the recipient) has performed a successful partial public key validation (see Section 5.6.2.6). For example, a trusted processor may only forward an ECC ephemeral public key to the recipient if it passes a partial public key validation.

Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

5.6.2.4 FFC Full Public Key Validation Routine

This method **shall** be used with static and ephemeral FFC public keys when assurance of the validity of the keys is obtained by method 1 or method 2 of Sections 5.6.2.1, 5.6.2.2, and 5.6.2.3.

If the IUT implements the FFC Full Public Key Validation Routine then it is tested by CAVS.

5.6.3.1 Owner Assurances of Possession of a Static Private Key

The owner of a static public key **shall** have assurance that the owner actually possesses the correct associated private key in one or more of the following ways:

1. Owner Receives Assurance via Explicit Key Confirmation – The owner employs the static key pair to successfully engage another party in a key agreement transaction incorporating explicit key confirmation. The key confirmation **shall** be performed with the owner as key confirmation recipient in order to obtain assurance that the private key functions correctly. See Section 8 for further explanation.
2. Owner Receives Assurance via Use of an Encrypted Certificate - The owner uses the static private key while engaging in a key agreement transaction with a Certificate Authority (trusted by the owner), providing the CA with the corresponding static public key. As part of this transaction, the CA generates a certificate containing the owner's static public key and encrypts the certificate using a symmetric key derived from the shared secret they have (allegedly) established. Only the encrypted form of the certificate is provided to the owner. By successfully decrypting the certificate, the owner obtains assurance of possession of the correct private key (at the time of the key agreement transaction).
3. Owner Receives Assurance via Key Regeneration – The owner regenerates a public key from the static private key and verifies that the regenerated public key is equal to the original static public key. Note that this method may be useful if the static private key has been generated by a party other than the owner or as an integrity check on a key pair that has been stored for a long period of time.
4. Owner Receives Assurance via Trusted Provision - A trusted party (trusted by the owner) provides the static private key and static public key to the owner using a trusted distribution method. Reliance upon this method assumes (1) that the trusted party will provide a private key that is consistent with the public key and (2) that the trusted party will not use the private key to masquerade as the owner.
5. Owner Receives Assurance via Key Generation - The act of generating a key pair, with the public key being computed from the private key, is a way for the owner to obtain assurance of possession of the correct private key. This method allows an owner who protects his/her own keys to have assurance of possession without additional computation. Note that this method may not detect algorithm implementation errors, hardware errors, random bit flips, etc. Further assurance may be obtained through the use of one or more of the above methods.

Out of scope of the algorithm testing. However, if the IUT has the supporting function key pair generation or key regeneration, the CAVS Validity Test tests this function.

5.6.4.1 Common Requirements on Static and Ephemeral Key Pairs

A public/private key pair **shall** be correctly associated with its corresponding specific set of domain parameters.

DSA/ECDSA Key Pair Generation is a prerequisite to KAS validation. CAVS testing gives correct and incorrect public/private key pairs to the IUT to see if it can detect erroneous key pair – domain parameter combinations.

2. Each DLC private key **shall** be unpredictable and created using an Approved key generation method as specified in Section 5.6.1.

RNG and/or DRBG are prerequisites to KAS validation.
DSA and/or ECDSA Key Pair Generation is a prerequisite to KAS validation.

5.7 DLC Primitives

Each scheme in Section 6 **shall** use an appropriate primitive from the following list:

1. The FFC DH primitive (Section 5.7.1.1 of this Recommendation): This primitive **shall** be used by the dhHybrid1, dhEphem, dhHybridOneFlow, dhOneFlow and dhStatic schemes, which are based on finite field cryptography and the Diffie-Hellman algorithm.
2. The ECC CDH primitive (Section 5.7.1.2 of this Recommendation and called the Modified Diffie-Hellman primitive in ANS X9.63): This primitive **shall** be used by the Full Unified Model, Ephemeral Unified Model, One-Pass Unified Model, One-Pass Diffie-Hellman and Static Unified Model schemes, which are based on elliptic curve cryptography and the Diffie-Hellman algorithm.
3. The FFC MQV primitive (Section 5.7.2.1 of this Recommendation): This primitive **shall** be used by the MQV2 and MQV1 schemes, which are based on finite field cryptography and the MQV algorithm.
4. The ECC MQV primitive (Section 5.7.2.2 of this Recommendation): This primitive **shall** be used by the Full MQV and One-Pass MQV schemes, which are based on elliptic curve cryptography and the MQV algorithm.

All of these "shall" statements are defining which DLC primitive is used by which schemes. The CAVP tests the DLC primitive when testing the schemes.

The shared secret output from these primitives **shall** be used as input to a key

derivation function (see Section 5.8).

The CAVS test uses the output from the shared secret as input to a key derivation function.

5.8 Key Derivation Functions for Key Agreement Schemes

An Approved key derivation function (KDF) **shall** be used to derive secret keying material from a shared secret.

If the IUT uses one of the 2 KDFS specified in SP800-56A, CAVS KAS testing tests the correctness of the KDF function. If the IUT uses a KDF not specified in SP800-56A, the CAVS Component test "All of 800-56A EXCEPT the KDF" can be performed.

If Key Confirmation (KC) or implementation validation testing are to be performed as specified in Section 8 or Section 5.2.3, respectively, then the MAC key **shall** be formed from the first bits of the KDF output …

The CAVS testing uses the first bits of the KDF as the MAC key to MAC the string in the testing. If the IUT is not using the same first bits of the KDF, the testing will indicate that the IUT is not implemented correctly.

Any hash function used in a KDF **shall** be Approved (see Section 5.1) and **shall** also meet the selection requirements specified herein (see Section 5.5.1).

Hash validation testing is a prerequisite to KAS validation.
CAVS testing only allows for the testing of specified parameter sets.

5.8.1 Concatenation Key Derivation Function (Approved Alternative 1)

keydatalen: An integer that indicates the length (in bits) of the secret keying material to be generated; *keydatalen* **shall** be less than or equal to *hashlen* × $(2^{32} - 1)$.

Implicitly tested by CAVS.

Any scheme attempting to call this key derivation function with *keydatalen* greater than or equal to *hashlen* × $(2^{32} - 1)$ **shall** output an error indicator and stop without outputting *DerivedKeyingMaterial*.

Implicitly tested by CAVS.

5.8.2 ASN.1 Key Derivation Function (Approved Alternative 2)

keydatalen: An integer that indicates the length (in bits) of the secret keying material to be generated; *keydatalen* **shall** be less than or equal to *hashlen* × (2^{32} –1).

Implicitly tested by CAVS.

Any call to this key derivation function using a *keydatalen* value that is greater than *hashlen* × (2^{32} –1) **shall** cause the KDF to output an error indicator and stop without outputting *DerivedKeyingMaterial*.

Implicitly tested by CAVS.

6. Key Agreement

Each party in a key agreement process **shall** use the same set of valid domain parameters.

CAVS testing requires both the IUT and CAVS to use the same domain parameters to successfully test an implementation.

Party U **shall** have an identifier ID_U.

CAVS testing requires Party U to supply an identifier.

If Party U owns a static key pair that is used in a given key agreement transaction, then ID_U **shall** be the identifier that is bound to that key pair.

CAVS testing requires the identifier and static key pair of the IUT and CAVS to successfully test an implementation.

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

All key pairs **shall** be generated using the same domain parameters.

CAVS testing requires both the static and ephemeral key pairs be generated using the same domain parameters.

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

> *CAVS testing requires both the static and ephemeral key pairs be generated using the same domain parameters.*

2. Each party **shall** have 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 (*ds, Qs*).

> *CAVS testing either assigns or requires the IUT and CAVS to supply the static key pair so implicitly they are being designated as the owner of that key.*

Each party **shall** obtain assurance of the validity of its own static public key as specified in Section 5.6.2.1.

> *Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.*

Each party **shall** obtain assurance of its possession of the correct value for its own private key as specified in Section 5.6.3.1.

> *Out of scope of the algorithm testing. However, if the IUT has the supporting function key generation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.*

3.The parties **shall** have agreed upon an Approved key derivation function (see Section 5.8) as well as an Approved hash function appropriate for use with the key derivation function and associated parameters (see Section 5.5).

> *CAVS testing requires the IUT to indicate what key derivation function and hash function it supports. CAVS then tests these functions with the KAS tests.*

If key confirmation is used, the parties **shall** have agreed upon an Approved MAC

and associated parameters (see Tables 1 and 2).

CAVS testing requires the IUT to indicate what MAC function it supports. CAVS then tests this function with the KAS tests.

MAC validation testing is a prerequisite to KAS validation.

Prior to or during the key agreement process, each party **shall** obtain the identifier associated with the other party during the key agreement scheme and the static public key that is bound to that identifier.

CAVS testing requires both parties to obtain the identifier associated with the other party during key agreement scheme and the static key bound to that identifier for the testing to succeed.

Each party **shall** obtain assurance of the validity of the other party's static public key as specified in Section 5.6.2.2.

Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.1.1. In particular, party U **shall** obtain the static public key *yV* of party V, and party V **shall** obtain the static public key *yU* of party U.
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 shared secret keying material from *Z*.
U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*rU, tU*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *tU* to V. Receive an ephemeral public key *tV* (purportedly) from V. If *tV* is not received, output an error indicator and stop.
2. Verify that *tV* 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, output an error indicator and stop.
3. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret *Zs* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's static private key *xU*, and V's static public key *yV*. Convert *Zs* to a byte string (which is also denoted by *Zs*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in

the computation of *Zs*. If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Zs*, output an error indicator, and stop.
4. Use the FCC DH primitive to derive a shared secret *Ze* – another integer in the range [2,*p*-2] – from the set of domain parameters *D*, U's ephemeral private key *rU*, and V's ephemeral public key *tV*. Convert *Ze* to a byte string (which is also denoted by *Ze*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Ze*. If this call to the FFC DH primitive outputs an error indicator, zeroize *Zs* and the results of all intermediate calculations used in the attempted computation of *Ze*, output an error indicator, and stop.
5. Compute the shared secret $Z = Ze \| Zs$. Zeroize the results of all intermediate calculations used in the computation of *Z* (including Z*e* and *Zs*).
6. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.

THIS APPLIES TO ALL OF SECTION 6.1.1.1

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

In particular, party U **shall** obtain the static public key*Qs,V* of party V, and party V **shall** obtain the static public key *Qs,U* of party U.
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 shared secret keying material from *Z*.
Actions: U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*de,U, Qe,U*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *Qe,U* to V. Receive an ephemeral public key *Qe,V*
(purportedly) from V. If *Qe,V* is not received, output an error indicator and stop.
2. Verify that *Qe,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, output an

error indicator and stop.
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Zs* – an element of the finite field of size *q* – from the set of domain parameters *D*, U's static private key *ds,U*, and V's static public key *Qs,V*. Convert *Zs* to a byte string (which is also denoted by *Zs*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Zs*. If the call to the ECC CDH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Zs*, output an error indicator, and stop.
4. Use the ECC CDH primitive to derive a shared secret *Ze* – another element of the finite field of size *q* – from the set of domain parameters D, U's ephemeral private key *de,U*, and V's ephemeral public key *Qe,V*. Convert *Ze* to a byte string (which is also denoted by *Ze*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Ze*. If this call to the ECC CDH primitive outputs an error indicator, zeroize *Zs* and the results of all intermediate calculations used in the attempted computation of *Ze*, output an error indicator, and stop.
5. Compute the shared secret $Z = Ze \| Zs$. Zeroize the results of all intermediate calculations used in the computation of *Z* (including Z*e* and *Zs*).
6. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
7. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.1.1.2

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.1.1. In particular, party U **shall** obtain the static public key *yV* of party V, and party V **shall** obtain the static public key *yU* of party U.

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 shared secret keying material from *Z*.
U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*rU, tU*) from the domain parameters *D* as specified in
Section 5.6.1.1. Send the public key *tU* to V. Receive an ephemeral public key *tV* (purportedly) from V. If *tV* is not received, output an error indicator and stop.
2. Verify that *tV* 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, output an error indicator and stop.
3. Use the MQV2 form of the FFC MQV primitive in Section 5.7.2.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's static private key *xU*, V's static public key *yV*, U's ephemeral private key *rU*, U's ephemeral public key *tU*, and V's ephemeral public key *tV*. If the call to the FFC MQV primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.1.1.3

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.1.1. In particular, party U **shall** obtain the static public key *Qs,V* of party V, and party V **shall** obtain the static public key *Qs,U* of party U.

Party *U* **shall** execute the following transformation to a) establish a shared secret value *Z* with party V, and b) derive shared secret keying material from *Z*.
Actions: U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*de,U*, *Qe,U*) from the domain parameters *D* as specified in Section 5.6.1.2. Send the public key *Qe,U* to V. Receive an ephemeral public key *Qe,V*
(purportedly) from V. If *Qe,V* is not received, output an error indicator and stop.
2. Verify that *Qe,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, output an error indicator and stop.
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* – an element of the finite field of size *q* – from the set of domain parameters *D*, U's static private key *ds,U*, V's static public key *Qs,V*, U's ephemeral private key *de,U*, U's ephemeral public key *Qe,U*, and V's ephemeral public key *Qe,V*. If the call to the ECC MQV primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-to-Byte String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.1.1.4

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

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

CAVS testing requires both the static and ephemeral key pairs be generated using the same domain parameters.

3.The parties **shall** have agreed upon an Approved key derivation function (see Section 5.8) as well as an Approved hash function appropriate for use with the key derivation function and associated parameters (see Section 5.5).

CAVS testing requires the IUT to indicate what key derivation function and hash function it supports. CAVS then tests these functions with the KAS tests.

If key confirmation is used, the parties **shall** have agreed upon an Approved MAC and associated parameters (see Tables 1 and 2).

MAC validation testing is a prerequisite to KAS validation.

4.Prior to or during the key agreement process, each party **shall** obtain the identifier associated with the other party during the key agreement scheme.

CAVS testing requires both parties to obtain the identifier associated with the other party during key agreement scheme and the static key bound to that identifier for the testing to succeed.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.1.2.
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 shared secret keying material from *Z.*
Actions: U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*rU, tU*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *tU* to V. Receive an ephemeral public key *tV* (purportedly) from V. If *tV* is not received, output an error indicator and stop.
2. Verify that *tV* 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, output an error indicator and stop.
3. Use the FCC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's ephemeral private key *rU,*
and V's ephemeral public key *tV.* If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the

attempted computation of *Z*, output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.

6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.1.2.1

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.1.2. 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 shared secret keying material from *Z*.
Actions: U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*de,U, Qe,U*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *Qe,U* to V. Receive an ephemeral public key *Qe,V* (purportedly) from V. If *Qe,V* is not received, output an error indicator and stop.
2. Verify that *Qe,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, output an error indicator and stop.
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* – an element of the finite field of size *q* – from the set of domain parameters D, U's ephemeral private key *de,U,* and V's ephemeral public key *Qe,V*. If the call to the ECC CDH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-

to-Byte- String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.1.2.2

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

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

> *CAVS testing requires both the static and ephemeral key pairs be generated using the same domain parameters.*

2. Each party **shall** have 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 (*ds, Qs*).

> *CAVS testing either assigns or requires the IUT and CAVS to supply the static key pair so implicitly they are being designated as the owner of that key.*

Each party **shall** obtain assurance of the validity of its own static public key as specified in Section 5.6.2.1.

Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

Each party **shall** obtain assurance of its possession of the correct value for its own private key as specified in Section 5.6.3.1.

Out of scope of the algorithm testing. However, if the IUT has the supporting function key generation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

3.The parties **shall** have agreed upon an Approved key derivation function (see Section 5.8) as well as an Approved hash function appropriate for use with the key derivation function and associated parameters (see Section 5.5).

CAVS testing requires the IUT to indicate what key derivation function and hash function it supports. CAVS then tests these functions with the KAS tests.

If key confirmation is used, the parties **shall** have agreed upon an Approved MAC and associated parameters (see Tables 1 and 2).

MAC validation testing is a prerequisite to KAS validation.

4. Prior to or during the key agreement process, each party **shall** obtain the identifier associated with the other party during the key agreement scheme and the static public key
that is bound to that identifier.

CAVS testing requires both parties to obtain the identifier associated with the other party during key agreement scheme and the static key bound to that identifier for the testing to succeed.

Each party **shall** obtain assurance of the validity of the other party's static public key as specified in Section 5.6.2.2.

Out of scope of the algorithm testing. However, if the IUT has the

supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.2.1. In particular, party U **shall** obtain the static public key *yV* of party V, and party V **shall** obtain the static public key *yU* of party U.

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 shared secret keying material from *Z*.

Actions: U **shall** derive secret keying material as follows:

1. Generate an ephemeral key pair (*rU, tU*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *tU* to V.
2. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret *Zs* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's static private key *xU*, and V's static public key *yV*. Convert *Zs* to a byte string (which is also denoted by *Zs*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Zs*. If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Zs*, output an error indicator, and stop.
3. Use the FCC DH primitive to derive a shared secret *Ze* – another integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's ephemeral private key *rU*, and V's static public key *yV*. Convert *Ze* to a byte string (which is also denoted by *Ze*) using the Integerto-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Ze*. If this call to the FFC DH
primitive outputs an error indicator, zeroize *Zs* and the results of all intermediate calculations used in the attempted computation of *Ze*, output an error indicator, and stop.
4. Compute the shared secret $Z = Ze \,||\, Zs$. Zeroize the results of all intermediate calculations used in the computation of *Z* (including Z*e* and *Zs*).
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

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

Actions: V **shall** derive secret keying material as follows:

1. Receive an ephemeral public key *tU* (purportedly) from U. If *tU* is not received, output an error indicator and stop.

2. Verify that *tU* 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, output an error indicator and stop.
3. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret *Zs* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, V's static private key *xV,* and U's static public key *yU.* Convert *Zs* to a byte string (which is also denoted by *Zs*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Zs*. If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Zs,* output an error indicator, and stop.
4. Use the FCC DH primitive to derive a shared secret *Ze* – another integer in the range [2,*p*-2] – from the set of domain parameters *D*, V's static private key *xV,* and U's ephemeral public key *tU.* Convert *Ze* to a byte string (which is also denoted by *Ze*) using the Integerto-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Ze*. If this call to the FFC DH primitive outputs an error indicator, zeroize *Zs* and the results of all intermediate calculations used in the attempted computation of *Ze,* output an error indicator, and stop.
5. Compute the shared secret *Z = Ze || Zs.* Zeroize the results of all intermediate calculations used in the computation of *Z* (including Z*e* and *Zs*).
6. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z,* output an error indicator, and stop.
7. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.2.1.1

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.2.1. In particular, party U **shall** obtain the static public key *Qs,V* of party V, and party V **shall** obtain the static public key *Qs,U* of party U.

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 shared secret keying material from *Z*.

Actions: U **shall** derive secret keying material as follows:

1. Generate an ephemeral key pair (*de,U, Qe,U*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *Qe,U* to V.
2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Zs* – an element of the finite field of size *q* – from the set of domain parameters *D*, U's static private key *ds,U,* and V's static public key *Qs,V.* Convert *Zs* to a byte string (which is also denoted by *Zs*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Zs.* If the call to the ECC CDH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Zs,* output an error indicator, and stop.
3. Use the ECC CDH primitive to derive a shared secret *Ze* – another element of the finite field of size *q* – from the set of domain parameters D, U's ephemeral private key *de,U,* and V's static public key *Qs,V.* Convert *Ze* to a byte string (which is also denoted by *Ze*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Ze*. If this call to the ECC CDH primitive outputs an error indicator, zeroize *Zs* and the results of all intermediate calculations used in the attempted computation of *Ze,* output an error indicator, and stop.
4. Compute the shared secret $Z = Ze \,||\, Zs$. Zeroize the results of all intermediate calculations used in the computation of *Z* (including *Ze* and *Zs*).
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z,* output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

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

Actions: V **shall** derive secret keying material as follows:

1. Receive an ephemeral public key *Qe,U* (purportedly) from U. If *Qe,U* is not received, output an error indicator and stop.
2. Verify that *Qe,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, output an error indicator and stop.
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Zs* – an element of the finite field of size *q* – from the set of domain parameters *D*, V's static private key *ds,V,* and U's static public key *Qs,U.* Convert *Zs* to a byte string (which is also denoted by *Zs*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate

calculations used in the computation of *Zs.* If the call to the ECC CDH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Zs,* output an error
indicator, and stop.
4. Use the ECC CDH primitive to derive a shared secret *Ze* – another element of the finite field of size *q* – from the set of domain parameters D, V's static private key *ds,V,* and U's ephemeral public key *Qe,U.* Convert *Ze* to a byte string (which is also denoted by *Ze*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Ze.* If this call to the ECC CDH primitive outputs an error indicator, zeroize *Zs* and the results of all intermediate calculations used in the attempted computation of *Ze,* output an error indicator, and stop.
5. Compute the shared secret *Z = Ze || Zs.* Zeroize the results of all intermediate calculations used in the computation of *Z* (including Z*e* and *Zs*).
6. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z,* output an error indicator, and stop.
7. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial.*

THIS APPLIES TO ALL OF SECTION 6.2.1.2

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

In particular, party U **shall** obtain the static public
key *yV* of party V, and party V **shall** obtain the static public key *yU* of party U. 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 shared secret keying material from *Z.*
Actions: U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*rU, tU*) from the domain parameters *D* as specified in
Section 5.6.1. Send the public key *tU* to V.

2. Use the MQV1 form of the FFC MQV primitive in Section 5.7.2.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's static private key *xU*, V's static public key *yV*, U's ephemeral private key *rU*, U's ephemeral public key *tU*, and (for a second time) V's static public key *yV*. If the call to the FFC MQV primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.
Party V **shall** execute the following key agreement transformation in order to a) establish a shared secret value *Z* with party U, and b) derive shared secret keying material from *Z*.
Actions: V **shall** derive secret keying material as follows:
1. Receive an ephemeral public key *tU* (purportedly) from U. If *tU* is not received, output an error indicator and stop.
2. Verify that *tU* 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, output an error indicator and stop.
3. Use the MQV1 form of the FFC MQV primitive in Section 5.7.2.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, V's static private key *xV*, U's static public key *yU*, V's static private key *xV* (for a second time), V's static public key *yV*, and U's ephemeral public key *tU*. If the call to the FFC MQV primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.2.1.3

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.2.1. In particular, party U **shall** obtain the static public key *Qs,V* of party V, and party V **shall** obtain the static public key *Qs,U* of party U.

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

Actions: U **shall** derive secret keying material as follows:

1. Generate an ephemeral key pair (*de,U, Qe,U*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *Qe,U* to V.
2. Use the One-Pass MQV form of the ECC MQV primitive in Section 5.7.2.3.2 to derive a shared secret value *Z* – an element of the finite field of size *q* – from the set of domain parameters *D*, U's static private key *ds,U,* V's static public key *Qs,V,* U's ephemeral private key *de,U,* U's ephemeral public key *Qe,U,* and (for a second time) V's static public key of all intermediate calculations used in the attempted computation of *Z,* output an error indicator, and stop.
3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-to-Byte String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z,* output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

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

Actions: V **shall** derive secret keying material as follows:

1. Receive an ephemeral public key *Qe,U* (purportedly) from U. If *Qe,U* is not received, output an error indicator and stop.
2. Verify that *Qe,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, output an error indicator and stop.
3. Use the One-Pass MQV form of the ECC MQV primitive in Section 5.7.2.3 to derive a shared secret value *Z* – an element of the finite field of size *q* – from the set of domain parameters *D*, V's static private key *ds,V,* U's static public key *Qs,U,* V's static private key *ds,*V (for a second time), V's static public key *Qs,V,*

and U's ephemeral public key *Qe,U*. If the call to the ECC MQV primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-to-Byte String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.2.1.4

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

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

CAVS testing requires both the static and ephemeral key pairs be generated using the same domain parameters.

2.The responder **shall** obtain assurance of the validity of its own static public key as specified in Section 5.6.2.1.

Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

The responder **shall** obtain assurance of its possession of the correct value of its own private key as specified in Section 5.6.3.1.

Out of scope of the algorithm testing. However, if the IUT has the supporting function key generation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

3.The parties **shall** have agreed upon an Approved key derivation function (see Section 5.8) as well as an Approved hash function appropriate for use with the key derivation function and associated parameters (see Section 5.5).

CAVS testing requires the IUT to indicate what key derivation function and hash function it supports. CAVS then tests these functions with the KAS tests.

If key confirmation is used, the parties **shall** have agreed upon an Approved MAC and associated parameters (see Tables 1 and 2).

MAC validation testing is a prerequisite to KAS validation.

4. Prior to or during the key agreement process, each party **shall** obtain the identifier associated with the other party during the key agreement scheme. The initiator **shall** obtain the static public key that is bound to the responder's identifier.

CAVS testing requires both parties to obtain the identifier associated with the other party during key agreement scheme and the static key bound to that identifier for the testing to succeed.

The initiator **shall** obtain assurance of the validity of the responder's static public key as specified in Section 5.6.2.2.

Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.2.2.

In particular, party U **shall** obtain the static public key *yV* of party V.
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 shared secret keying material from *Z*.
Actions: U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*rU, tU*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *tU* to V.
2. Use the FCC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's ephemeral private key *rU*, and V's static public key *yV*. If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.
Party V **shall** execute the following key agreement transformation in order to a) establish a shared secret value *Z* with party U, and b) derive shared secret keying material from *Z*.
Actions: V **shall** derive secret keying material as follows:
1. Receive an ephemeral public key *tU* (purportedly) from U. If *tU* is not received, output an error indicator and stop.
2. Verify that *tU* 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, output an error indicator and stop.
3. Use the FCC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, V's static private key *xV*, and U's ephemeral public key *tU*. If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.2.2.1

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.2.2. In particular, party U **shall** obtain the static public key *Qs,V* of party V.
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 shared secret keying material from *Z*.
Actions: U **shall** derive secret keying material as follows:
1. Generate an ephemeral key pair (*de,U, Qe,U*) from the domain parameters *D* as specified in Section 5.6.1. Send the public key *Qe,U* to V.
2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* – an element of the finite field of size *q* – from the set of domain parameters *D*, U's ephemeral private key *de,U*, and V's static public key *Qs,V*. If this call to the ECC CDH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.
Party V **shall** execute the following key agreement transformation in order to a) establish a shared secret value *Z* with party U, and b) derive shared secret keying material from *Z*.
Actions: V **shall** derive secret keying material as follows:
1. Receive an ephemeral public key *Qe,U* (purportedly) from U. If *Qe,U* is not received, output an error indicator and stop.
2. Verify that *Qe,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, output an

error indicator and stop.
3. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* – an element of the finite field of size *q* – from the set of domain parameters *D*, V's static private key *ds,V,* and U's ephemeral public key *Qe,U.* If this call to the ECC CDH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z,* output an error indicator, and stop.
4. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Z.*
5. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*). (See Section 5.8.) If the key derivation function outputs an error indicator, zeroize all copies of *Z,* output an error indicator, and stop.
6. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.2.2.2

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string.

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

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

CAVS testing requires both the static and ephemeral key pairs be generated using the same domain parameters.

2. Each party **shall** have 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 (*ds, Qs*).

CAVS testing either assigns or requires the IUT and CAVS to supply the static key pair so implicitly they are being designated as the owner of that key.

Each party **shall** obtain assurance of the validity of its own static public key as specified in Section 5.6.2.1.

Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

Each party **shall** obtain assurance of its possession of the correct value for its own private key as specified in Section 5.6.3.1.

Out of scope of the algorithm testing. However, if the IUT has the supporting function key generation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.

3.The parties **shall** have agreed upon an Approved key derivation function (see Section 5.8) as well as an Approved hash function appropriate for use with the key derivation function and associated parameters (see Section 5.5).

CAVS testing requires the IUT to indicate what key derivation function and hash function it supports. CAVS then tests these functions with the KAS tests.

If key confirmation is used, the parties **shall** have agreed upon an Approved MAC and associated parameters (see Tables 1 and 2).

MAC validation testing is a prerequisite to KAS validation.

4. Prior to or during the key agreement process, each party **shall** obtain the identifier associated with the other party during the key agreement scheme and the static public key that is bound to that identifier.

CAVS testing requires both parties to obtain the identifier associated with the other party during key agreement scheme and the static key bound to that identifier for the testing to succeed.

Each party **shall** obtain assurance of the validity of the other party's static public key as specified in Section 5.6.2.2.

> *Out of scope of the algorithm testing. However, if the IUT has the supporting function full and/or partial public key validation, the CAVS Validity Test tests that the IUT can detect an erroneous public/private key pair.*

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.3. In particular, party U **shall** obtain the static public key y_V of party V, and party V **shall** obtain the static public key y_U of party U.

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 shared secret keying material from *Z*.

Actions: U **shall** derive secret keying material as follows:

1. Obtain a nonce, *NonceU* (see Section 5.4). Send *NonceU* to V.
2. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, U's static private key *xU*, and V's static public key *yV*. If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers IDu and IDv, and *NonceU*.). *NonceU* **shall** be in the *PartyUInfo* subfield of *OtherInfo*. If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

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

Actions: V **shall** derive secret keying material as follows:

1. Obtain U's nonce, *NonceU*, from U. If *NonceU* is not available, output an error indicator
and stop.
2. Use the FFC DH primitive in Section 5.7.1.1 to derive a shared secret *Z* – an integer in the range [2, *p*-2] – from the set of domain parameters *D*, V's static private key *xV*, and U's static public key *yU*. If the call to the FFC DH primitive outputs an error indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.

3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Integer-to-Byte-String Conversion specified in Appendix C.1, and then zeroize the results of all intermediate calculations used in the computation of *Z*.
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*, and *NonceU*). *NonceU* **shall** be in the *PartyUInfo* subfield of *OtherInfo.* If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.3.1

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string. (Note NonceU is looked for.)

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

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

The prerequisites for this scheme **shall** be satisfied as specified in Section 6.3. In particular, party U **shall** obtain the static public key *Qs,V* of party V, and party V **shall** obtain the static public key *Qs,U* of party U.
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 shared secret keying material from *Z*.
Actions: U **shall** derive secret keying material as follows:
1. Obtain a nonce, *NonceU* (see Section 5.4). Send *NonceU* to V.
2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* – an element of the finite field of size *q* – from the set of domain parameters *D*, U's static private key *ds,U, and* V's static public key *Qs,V.* If the call to the ECC CDH primitive outputs an error
indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all
intermediate calculations used in the computation of *Z.*
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers *IDU* and *IDV*, and *NonceU*). *NonceU*

shall be in the *PartyUInfo sub*field of *OtherInfo.* If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.
Party V **shall** execute the following key agreement transformation in order to a) establish a shared secret value, *Z*, with party U, and b) derive shared secret keying material from *Z*.
1. Obtain U's nonce, *NonceU*, from U. If *NonceU* is not if available, output an error indicator and stop.
2. Use the ECC CDH primitive in Section 5.7.1.2 to derive a shared secret *Z* – an element of the finite field of size *q* – from the set of domain parameters *D*, V's static private key *ds,V*, and U's static public key *Qs,U*. If the call to the ECC CDH primitive outputs an error
indicator, zeroize the results of all intermediate calculations used in the attempted computation of *Z*, output an error indicator, and stop.
3. Convert *Z* to a byte string (which is also denoted by *Z*) using the Field-element-to-Byte-String Conversion specified in Appendix C.2, and then zeroize the results of all intermediate calculations used in the computation of *Zs.*
4. Use the agreed-upon key derivation function to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value *Z* and *OtherInput* (including the identifiers IDu and IDv, and *NonceU*.). *NonceU* **shall** be in the *PartyUInfo* subfield of *OtherInfo*. If the key derivation function outputs an error indicator, zeroize all copies of *Z*, output an error indicator, and stop.
5. Zeroize all copies of the shared secret *Z* and output *DerivedKeyingMaterial*.

THIS APPLIES TO ALL OF SECTION 6.3.2

Tested by Validity Test and Function Test with the following exceptions:

Testing of zeroized values is out of scope

Testing of format of OtherInput field is out of scope. IDu and IDv are not looked for in OtherInput string. (Note, NonceU is looked for.)

Assurance of public key validity is out of scope of the algorithm testing. However, if the IUT has the supporting function public key validation, the CAVS Validity Test tests this function.

8. Key Confirmation

For key confirmation to comply with this Recommendation, key confirmation **shall** be incorporated into key establishment schemes as specified in this section.

If an IUT supports key confirmation, then it is tested by CAVS using the specification in this section.

If key confirmation is incorporated into a scheme in which a recipient does not provide an ephemeral public key, a nonce **shall** be provided for the key confirmation process.

CAVP – tests this scheme. It requires V to supply a nonce.

8.1 Assurance of Possession Considerations when using Key Confirmation

The key agreement scheme (including the key confirmation) **shall** be performed as described in this Recommendation.

See next section.

8.2 Unilateral Key Confirmation for Key Agreement Schemes

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.
1. If the recipient does not have an ephemeral key pair and has not already provided a nonce as part of the scheme, then the recipient **shall** provide a nonce to be used in its place (see Section 5.4).

CAVP – tests this scheme. It requires U to supply a nonce.

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

Since V does not contribute an ephemeral public key during the key agreement process, a nonce (*NonceV*) **shall** be provided to U prior to the computation of the *MacTag* and used as the *EphemDataV* during *MacTag* computations.

CAVP – tests this scheme. It requires U to supply a nonce.

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

V **shall** contribute a nonce (*NonceV*) prior to U's computation of the *MacTagU*.

CAVP – tests this scheme. It requires V to supply a nonce.

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

V **shall** contribute a nonce (*NonceV*) to U prior to the generation of the *MacTagu.*

CAVP – tests this scheme. It requires V to supply a nonce.

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

V **shall** contribute a nonce (*NonceV*) prior to the generation of *MacTagU*.

CAVP – tests this scheme. It requires V to supply a nonce.

10. Implementation Validation

When the NIST CMVP has established a validation program for this Recommendation, a vendor **shall** have its implementation tested and validated by the CMVP in order to claim conformance to this Recommendation.

CAVS validation testing provides a method to achieve this requirement.

An implementation claiming conformance to this Recommendation **shall** include one or more of the following capabilities:
• Domain parameter generation as specified in Section 5.5.1.
• A key agreement scheme from Section 6, together with an Approved key derivation function from Section 5.8. Other key derivation methods may be temporarily allowed for backward compatibility. These other allowable methods and any restrictions on their use will be specified in FIPS 140-2 Annex D. If key confirmation is also claimed, the
appropriate key confirmation technique from Section 8 **shall** be used.

At least one key agreement scheme must be implemented by an IUT for CAVS to test it.

An implementer **shall** also identify the appropriate specifics of the implementation, including:
• The security strength(s) of supported cryptographic algorithms; this will determine the
parameter set requirements (see Tables 1 and 2 in Section 5.5.1),
• The domain parameter generation method (see Section 5.5.1).
• The hash function (see Section 5.1),
• The MAC key size(s) (see Tables 1 and 2 in Section 5.5.1),
• The MAC length(s) (see Tables 1 and 2 in Section 5.5.1),
• The type of cryptography: FFC or ECC,
• The key establishment schemes available (see Section 6),
• The key derivation function to be used, including the format of *OtherInfo* (see Section

5.8),
• The type of nonces to be generated (see Section 5.4),
• The NIST Recommended elliptic curve(s) available (if appropriate), and
• The key confirmation scheme (see Section 8).

CAVS requires the IUT to supply this information.

Appendix C: Data Conversions (Normative)

The bytes of *S* **shall** satisfy:

$C = \Sigma\ 28(n\text{-}i)Si$ for $i = 1$ to n.

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.
The rightmost bit *sm* **shall** become the rightmost bit of the last byte *Sn*, and so on through
the leftmost bit *s*1, which **shall** become the (8*n* - *m* + 1)th bit of the first byte *S*1. The
leftmost (8*n* - *m*) bits of the first byte *S*1 **shall** be zero.

If $q = 2m$, then α must be a bit string of length *m* bits. Let *s*1, *s*2, …, *sm* be the bits of α

from leftmost to rightmost. α **shall** be converted to an integer *x* satisfying:

$x = \Sigma\ 2^{(m-i)}\, si$ for $i = 1$ to m.

CAVS supports this.