Suite B Implementer's Guide to NIST SP 800-56A

July 28, 2009

1. Introduction

This document specifies the Elliptic Curve Diffie-Hellman (ECDH) key-agreement schemes from NIST SP 800-56A: "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography" that will be used in future and existing cryptographic protocols for Suite B products. Also included are the elliptic curves and domain parameters, key generation methods, the ECDH primitive, key derivation function, and other auxiliary functions that are necessary for ECDH scheme implementations to be in compliance with SP 800-56A and Suite B.

Several IETF protocols (e.g., IKE/IPsec, S/MIME, SSH, and TLS) were designed and gained widespread use prior to the publication of SP 800-56A. These protocols do not comply with all of the key-agreement requirements specified in SP 800-56A. Discussion concerning the implementation of ECDH in these protocols for Suite B products is also given in this document.

2. Definitions, Symbols and Abbreviations[1]

2.1 Definitions

Approved	FIPS approved or NIST Recommended. An algorithm or technique that is either 1) specified in a FIPS or NIST Recommendation, or 2) adopted in a FIPS or NIST Recommendation and specified either (a) in an appendix to the FIPS or NIST Recommendation, or (b) in a document referenced by the FIPS or NIST Recommendation
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.
Bit length	The length in bits of a bit string.

[1] These definitions, symbols and abbreviations are from NIST SP-800-56A. NIST SP-800-56A definitions, symbols and abbreviations that are not used in this Suite B Implementer's Guide have been omitted. In a few cases, clarifications have been added that are specific to Suite B.

Certification Authority	The entity in a Public Key Infrastructure (PKI) that is responsible for issuing public key certificates and exacting compliance to a PKI policy.
Cofactor	The order of the elliptic curve group divided by the (prime) order of the generator point specified in the domain parameters. Note that for the Suite B domain parameters, this cofactor is one.
Domain parameters	The parameters used with a cryptographic algorithm that are common to a domain of users.
Entity	An individual (person), organization, device, or process. "Party" is a synonym.
Ephemeral key	A key that is intended for a very short period of use. The key is ordinarily used in exactly one transaction of a cryptographic scheme; an exception to this is when the ephemeral key is used in multiple transactions for a key transport broadcast . Contrast with static key.
Hash function	A function that maps a bit string of arbitrary length to a fixed length bit string. Approved hash functions satisfy the following properties: 1. (One-way) It is computationally infeasible to find any input that maps to any pre-specified output, and 2. (Collision resistant) It is computationally infeasible to find any two distinct inputs that map to the same output. Approved hash functions for Suite B are SHA-256 and SHA-384, specified in FIPS 180-3.
Initiator	The party that begins a key-agreement transaction. Contrast with responder.
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. Contrast with key transport.[2]
Key-agreement transaction	The instance that results in shared secret keying material among different parties using a key-agreement scheme.

[2] Key transport algorithms are not used in Suite B; hence the term "key transport" is not included in the definitions section. Interested readers should consult NIST SP-800-56A.

Key confirmation	A procedure to provide assurance to one party (the key-confirmation recipient) that another party (the key-confirmation provider) actually possesses the correct secret keying material and/or shared secret.
Key derivation	The process by which keying material is derived from a shared secret and other information.
Key-establishment	The procedure that results in shared secret keying material among different parties.
Key-establishment transaction	An instance of establishing secret keying material using a key-establishment scheme.
Keying material	The data that is necessary to establish and maintain a cryptographic keying relationship. Some keying material may be secret, while other keying material may be public. As used in this document, secret keying material may include keys, secret initialization vectors or other secret information; public keying material includes any non-secret data needed to establish a relationship.
MacTag	Data that allows an entity to verify the integrity of the information. Other documents sometimes refer to this data as a MAC.
Message Authentication Code (MAC) algorithm	Defines a family of one-way cryptographic functions that is parameterized by a symmetric key and produces a *MacTa*g on arbitrary data. A MAC algorithm can be used to provide data origin authentication as well as data integrity. In this document, a MAC algorithm is used for key confirmation and validation testing purposes.
Owner	For a static key pair, the owner is the entity that is authorized to use the static private key associated with a public key, whether that entity generated the static key pair itself or a trusted party generated the key pair for the entity. For an ephemeral key pair, the owner is the entity that generated the key pair.
Party	An individual (person), organization, device, or process. “Entity” is a synonym for party.
Provider	The party during key confirmation that provides assurance to the other party (the recipient) that the two parties have indeed established a shared secret.

Public key certificate	A set of data that contains an entity's identifier(s), the entity's public key (including an indication of the associated set of domain parameters) and possibly other information, and is digitally signed by a trusted party, thereby binding the public key to the included identifier(s).
Recipient	A party that receives (1) keying material: such as a static public key (e.g., in a certificate) or an ephemeral public key; (2) assurance: such as an assurance of the validity of a candidate public key or assurance of possession of the private key associated with a public key; or (3) key confirmation. Contrast with provider.
Responder	The party that does not begin a key-agreement transaction. Contrast with initiator.
Scheme	A (cryptographic) scheme consists of an unambiguous specification of a set of transformations that are capable of providing a (cryptographic) service when properly implemented and maintained. A scheme is a higher level construct than a primitive and a lower level construct than a protocol.
Security strength (also "bits of security")	A number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system.
Shared secret keying material	The secret keying material that is either (1) derived by applying the key derivation function to the shared secret and other shared information during a key-agreement process, or (2) is transported during a key transport process[3].
Shared secret	A secret value that has been computed using a key-agreement scheme and is used as input to a key derivation function.
Static key	A key that is intended for use for a relatively long period of time and is typically intended for use in many instances of a cryptographic key-establishment scheme. Contrast with an ephemeral key.
Symmetric key algorithm	A cryptographic algorithm that uses one secret key that is shared between authorized parties.

[3] Key transport algorithms are not used in Suite B; hence the term "key transport" is not included in the definitions section. Interested readers should consult NIST SP-800-56A.

Trusted party	A trusted party is a party that is trusted by an entity to faithfully perform certain services for that entity. An entity may choose to act as a trusted party for itself.
Trusted third party	A third party, such as a CA, that is trusted by its clients to perform certain services. (By contrast, the initiator and responder in a scheme are considered to be the first and second parties in a key-establishment transaction.

2.2 Symbols and Abbreviations

General:

AES	Advanced Encryption Standard (as specified in FIPS 197)
ANS	American National Standard
ASN.1	Abstract Syntax Notation One
CA	Certification Authority
CDH	The cofactor Diffie-Hellman key-agreement primitive.
EC	Elliptic Curve.
ECC	Elliptic Curve Cryptography, the public key cryptographic methods using an elliptic curve. For example, see ANS X9.63.
HMAC	Keyed-Hash Message Authentication Code (as specified in FIPS 198).
ID	The bit string denoting the identifier associated with an entity.
H	An Approved hash function.
KC	Key Confirmation
KDF	Key Derivation Function
MAC	Message Authentication Code
Null	The empty bit string
SHA	Secure Hash Algorithm
TTP	A Trusted Third Party
U	The initiator of a key-establishment process.

V	The responder in a key-establishment process.
$\{X\}$	Indicates that the inclusion of X is optional.
$X \| \| Y$	Concatenation of two strings X and Y.
$\|x\|$	The length of x in bits.
$[a, b]$	The set of integers x such that $a \leq x \leq b$.
$\lceil x \rceil$	The ceiling of x; the smallest integer x. For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$.

ECC (ANS X9.63)

a, b	An ECC domain parameter; two field elements that define the equation of an elliptic curve.
$d_{e,U}$, $d_{e,V}$	Party U's and Party V's ephemeral private keys. These are integers in the range [1, n-1].
$d_{s,U}$, $d_{s,V}$	Party U's and Party V's static private keys. These are integers in the range [1, n-1].
D	The set of ECC domain parameters, (q, FR, a, b{, $SEED$}, G, n, h).
FR	Field Representation indicator. An indication of the basis used for representing field elements. For the Suite B curves, FR is NULL.
G	An ECC domain parameter, which is a distinguished point on an elliptic curve that generates the subgroup of order n.
h	An ECC domain parameter, the cofactor, which is the order of the elliptic curve divided by the order of the point G. For the Suite B curves, $h = 1$.
n	An ECC domain parameter; the order of the point G.
O	The point at infinity; a special point in an elliptic curve group that serves as the (additive) identity.
q	An ECC domain parameter; the field size.
$Q_{e,U}$, $Q_{e,V}$	Party U's and Party V's ephemeral public keys. These are points on the elliptic curve defined by the domain parameters.
$Q_{s,U}$, $Q_{s,V}$	Party U's and Party V's static public keys. These are points on the elliptic curve defined by the domain parameters.

SEED	An ECC domain parameter; an initialization value that is used during domain parameter generation that can also be used to provide assurance at a later time that the resulting domain parameters were generated arbitrarily.
x_p, y_p	Elements of the finite field of size q, representing the x and y coordinates respectively, of a point P. For Suite B curves, these are integers in the interval [0, q-1].
Z	A shared secret that is used to derive secret keying material using a key derivation function.
$Z_{bytestring}$	The byte-string form of Z. Note that this is not an ANS X9.63 term and was not used in SP 800-56A. It is introduced to distinguish between the field element Z and representation of Z in byte-string form.

3. ECDH Schemes

This section specifies the ECDH key-agreement schemes that can be used by Suite B products. The preferred ECDH scheme is the Ephemeral Unified Model (section 3.1), where each party generates an ephemeral key pair to be used in the computation of the shared secret. If the Ephemeral Unified Model cannot be used (for example, in a store-and-forward scenario where one party is not available to contribute an ephemeral public key), then the One-Pass Diffie-Hellman scheme is to be used. One-Pass Diffie-Hellman (section 3.2) is a key-agreement scheme in which an ephemeral key pair generated by one party is used together with the other party's static key pair in the computation of the shared secret.

It is important to note that, in practice, a key-agreement scheme is just one component of a larger (key-agreement) protocol, which may include many additional actions by both parties. Other components of the protocol may provide security services that are not provided by the key-agreement scheme itself. For example, when required, authentication of the source and integrity of exchanged ephemeral public keys must be provided by other components of a protocol incorporating either of these schemes.

Implementers should be aware that a number of protocols (e.g., IKE/IPsec, S/MIME, SSH and TLS) that were in widespread use prior to the publication of SP 800-56A do not comply with all of the key-agreement requirements specified below. In particular, alternate methods of key derivation have been defined for use in certain protocols. Section 7.1 of the "Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program" describes the deviations from the requirements of SP 800-56A that are currently allowed in products submitted for FIPS 140-2 validation. Some of the current waivers have expiration dates, so this document must be consulted for the most up-to-date information. Section 8 of the "Suite B Implementer's Guide to SP 800-

56A" contains further discussion regarding the IETF protocols IKE/IPsec, S/MIME, SSH and TLS for Suite B.

Protocols that use X.509 public key infrastructure certificates must follow the "Suite B Certificate and Certificate Revocation List Profile." This profile is a refinement of RFC 5280, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," and assumes that the reader is familiar with RFC 5280.

Certificate management follows the "Suite B Profile of Certificate Management over CMS[4]."

Prior to executing a routine that performs elliptic-curve computations on either an ephemeral or static public key, it is recommended that the party performing the computation should have assurance of the validity of that public key. The ECC Partial Public-Key Validation Routine (given in appendix B.3) can be employed whenever such assurance is required by Suite B compliant protocols.

Note: In the case of a static public key, the party seeking assurance may be in a position to trust that the CA established the subject public key's validity before issuing its certificate.

3.1 Ephemeral Unified Model

This section describes the Ephemeral Unified Model scheme from SP 800-56A. Using the same domain parameters, each party generates an ephemeral key pair. The two parties exchange ephemeral public keys and then compute the shared secret. The secret keying material is derived using the shared secret.

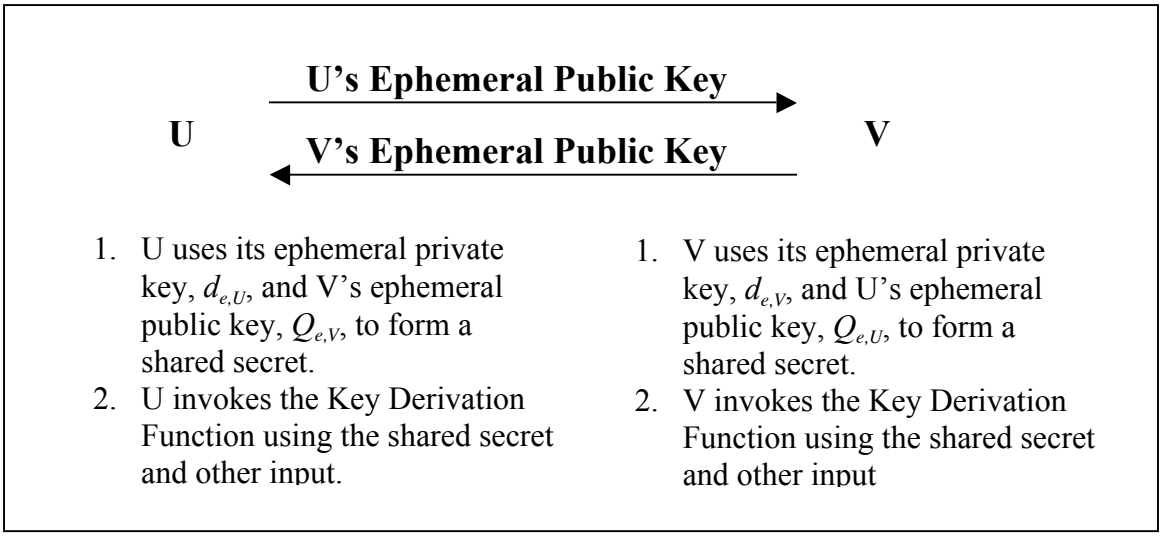

Figure 1: Each Party Generates an Ephemeral Key Pair; No Static Keys are Used.

[4] Cryptographic Message Syntax

Prerequisites:

1. Each party shall have an authentic copy of the same set of domain parameters, *D,* where $D = (q, FR, a, b\{, SEED\}, G, n, h)$. *D* must be selected from one of the two sets of domain parameters specified in Appendix A.
2. Each party shall use the NIST Concatenation Key Derivation Function[5] (see section 5). SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384 (see FIPS 180-3).
3. Prior to or during the key-agreement process, each party shall obtain the identifier associated with the other party during the key-agreement scheme.

With the exception of key derivation, Ephemeral Unified Model is "symmetric" in the actions of the initiator (Party U) and the responder (Party V).

Note, that U and V must use identical orderings of the bit strings that are input to the key derivation function in order for each party to produce the same secret keying material.

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 $(d_{e,U}, Q_{e,U})$ from the domain parameters *D* as specified in appendix B.1 or B.2. Send the public key $Q_{e,U}$ to V. Receive an ephemeral public key $Q_{e,V}$ (purportedly) from V. If $Q_{e,V}$ is not received, output an error indicator and stop.
2. Verify that $Q_{e,V}$ is a valid public key for the domain parameters *D* as specified in appendix B.3. If assurance of public key validity cannot be obtained, output an error indicator and stop.
3. Use the ECC CDH primitive specified in section 4 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 $d_{e,U}$ and V's ephemeral public key $Q_{e,V}$. 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 denoted by $Z_{bytestring}$[6]) using the Field-Element-to-Byte-String conversion specified in (see appendix C.3), and then zeroize the results of all intermediate calculations used in the computation of *Z*. The length of $Z_{bytestring}$ will be 32 bytes for P-256 and 48 bytes for P-384.

[5] Exceptions have been granted by NIST for certain IETF protocols. See NIST's "Implementation Guidance for FIPS 140-2 and the Cryptologic Module Validation Program" for the most current information regarding waivers for the KDFs used by particular IETF protocols.

[6] This term does not exist in SP 800-56A. SP 800-56A uses *Z* to denote both quantities: the *Z* output from the ECC CDH primitive and the byte-string form of *Z*.

5. Use the agreed upon key derivation function[7] to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value $Z_{bytestring}$ and *OtherInput* (including the identifiers ID_U and ID_V). If the key derivation function outputs an error indicator, zeroize all copies of Z and $Z_{bytestring}$, output an error indicator, and stop.
6. Zeroize all copies of the shared secret Z and $Z_{bytestring}$. Output the secret derived keying material *DerivedKeyingMaterial* or an error indicator.

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

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. Generate an ephemeral key pair ($d_{e,V}$, $Q_{e,V}$) from the domain parameters D as specified in appendix B.1 or B.2. Send the public key $Q_{e,V}$ to U. Receive an ephemeral public key $Q_{e,U}$ (purportedly) from U. If $Q_{e,U}$ is not received, output an error indicator and stop.
2. Verify that $Q_{e,U}$ is a valid public key for the domain parameters D as specified in appendix B.3. If assurance of public key validity cannot be obtained, output an error indicator and stop.
3. Use the ECC CDH primitive specified in section 4 to derive a shared secret Z – an element of the finite field of size q – from the set of domain parameters D, V's ephemeral private key $d_{e,V}$ and U's ephemeral public key $Q_{e,U}$. 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.
4. Convert Z to a byte string (which is denoted by $Z_{bytestring}$[8]) using the Field-Element-to-Byte-String conversion specified in (see appendix C.3), and then zeroize the results of all intermediate calculations used in the computation of Z. The length of $Z_{bytestring}$ will be 32 bytes for P-256 and 48 bytes for P-384.
5. Use the agreed upon key derivation function[9] to derive shared secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value $Z_{bytestring}$ and *OtherInput* (including the identifiers ID_U and ID_V). If the key derivation function outputs an error indicator, zeroize all copies of Z and $Z_{bytestring}$, output an error indicator, and stop.

---

[7] See section 5 for the NIST Concatenation Key Derivation Function. See section 8 and the referenced IETF RFCs for Suite B key derivation functions for the IETF protocols IKE/IPSec, S/MIME, SSH, and TLS.

[8] This term does not exist in SP 800-56A. SP 800-56A uses Z to denote both quantities: the Z output from the ECC CDH primitive and the byte-string form of Z.

[9] See section 5 for the NIST Concatenation Key Derivation Function. See section 8 and the referenced IETF RFCs for Suite B key derivation functions for the IETF protocols IKE/IPSec, S/MIME, SSH, and TLS.

6. Zeroize all copies of the shared secret Z and $Z_{bytestring}$. Output the secret derived keying material *DerivedKeyingMaterial* or an error indicator.

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

Table 1: Ephemeral Unified Model Key-agreement Scheme Summary

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

3.2 One-Pass Diffie-Hellman

This section describes the One-Pass Diffie-Hellman scheme from SP 800-56A. For this scheme, Party U generates an ephemeral key pair; Party V has only a static key pair. Party U obtains Party V's static public key in a trusted manner (for example, from a certificate signed by a trusted CA) and sends its ephemeral public key to Party V. Each party computes the shared secret by using its own private key and the other party's public key. Then each party uses the shared secret to derive shared secret keying material.

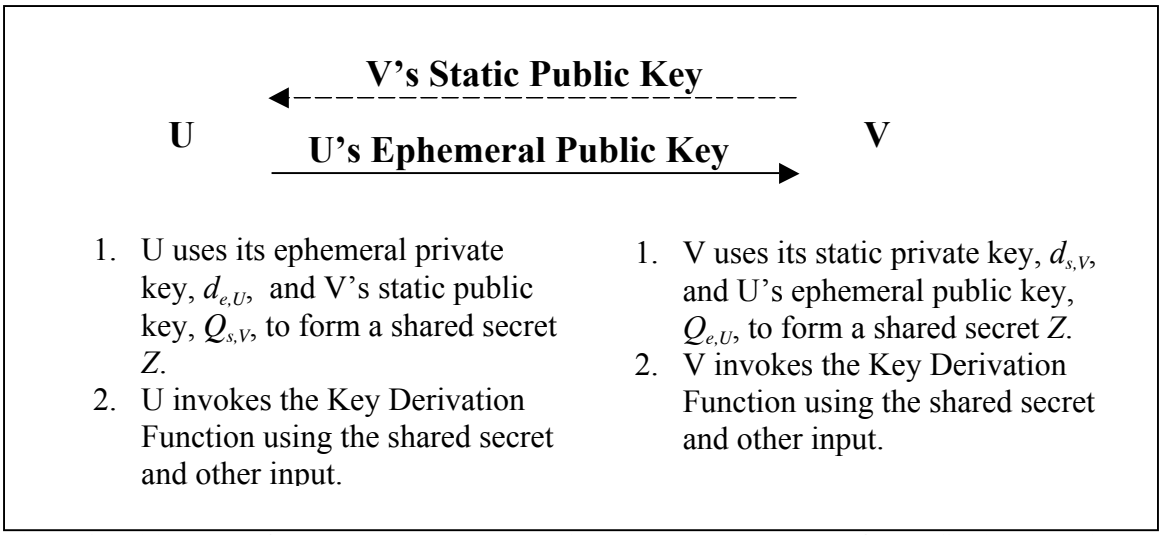

Figure 2: Initiator has Only an Ephemeral Key Pair and the Responder has Only a Static Key Pair.

Prerequisites:

1. Each party shall have an authentic copy of the same set of domain parameters, *D,* where $D = (q, FR, a, b\{, SEED\}, G, n, h)$. *D* must be selected from one of the two sets of domain parameters specified in Appendix A.
2. The responder shall have been designated as the owner of a static key pair that was generated as specified in appendix B, using the set of domain parameters, *D*.
3. The parties shall use the NIST Concatenation Key Derivation Function[10] in section 5. SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384 (see FIPS 180-3). If SP 800-56A-compliant key confirmation is used, the parties shall have agreed upon an Approved MAC and associated parameters (see section 6 for further guidance).
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. This static public key shall be obtained in a trusted manner (e.g., from a certificate signed by a trusted CA). The initiator shall obtain assurance of the validity of the responder's static public key has been established as specified in appendix B, using the set of domain parameters, *D*.

Note that U and V must use identical orderings of the bit strings that are input to the key derivation function in order for each party to produce the same shared secret keying material.

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*.

[10] Exceptions have been granted by NIST for certain IETF protocols. See NIST's "Implementation Guidance for FIPS 140-2 and the Cryptologic Module Validation Program" for the most current information regarding waivers for the KDFs used by particular IETF protocols.

Actions: U shall derive secret keying material as follows:

1. Generate an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) from the domain parameters D as specified in appendix B.1 or B.2. Send the public key $Q_{e,U}$ to V.
2. Use the ECC CDH primitive in section 4 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 $d_{e,U}$ and V's static public key $Q_{s,V}$. 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 denoted by $Z_{bytestring}$[11]) using the Field-Element-to-Byte-String Conversion specified in (see appendix C.3), and then zeroize the results of all intermediate calculations used in the computation of Z and $Z_{bytestring}$. The length of $Z_{bytestring}$ will be 32 bytes for P-256 and 48 bytes for P-384.
4. Use the agreed upon key derivation function[12] to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value $Z_{bytestring}$ and *OtherInput* (including the identifiers ID_U and ID_V).(See section 5). If the key derivation function outputs an error indicator, zeroize all copies of Z and $Z_{bytestring}$, output an error indicator, and stop.
5. Zeroize all copies of the shared secret Z and $Z_{bytestring}$. Output the secret derived keying material or an error indicator.
6. If SP 800-56A-compliant key confirmation is required, see section 6.

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

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 $Q_{e,U}$ (purportedly) from U. If $Q_{e,U}$ is not received, output an error indicator and stop.
2. Verify that $Q_{e,U}$ is a valid public key for the parameters D as specified in appendix B.3. If assurance of public key validity cannot be obtained, output an error indicator and stop.
3. Use the ECC CDH primitive in section 4 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 $d_{s,V}$ and U's ephemeral public key $Q_{e,U}$. If this call to the ECC CDH

[11] This term does not exist in SP 800-56A. SP 800-56A uses Z to denote both quantities: the Z output from the ECC CDH primitive and the byte-string form of Z.

[12] See section 5 for the NIST Concatenation Key Derivation Function. See section 8 and the referenced IETF RFCs for Suite B key derivation functions for the IETF protocols IKE/IPSec, S/MIME, SSH, and TLS.

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 denoted by $Z_{bytestring}$[13]) using the Field-Element-to-Byte-String Conversion specified in (see appendix C.3), and then zeroize the results of all intermediate calculations used in the computation of Z and $Z_{bytestring}$. The length of $Z_{bytestring}$ will be 32 bytes for P-256 and 48 bytes for P-384.
5. Use the agreed upon key derivation function[14] to derive secret keying material *DerivedKeyingMaterial* of length *keydatalen* bits from the shared secret value $Z_{bytestring}$ and *OtherInput* (including the identifiers ID_U and ID_V).(See section 5). If the key derivation function outputs an error indicator, zeroize all copies of Z and $Z_{bytestring}$, output an error indicator, and stop.
6. Zeroize all copies of the shared secret Z and $Z_{bytestring}$. Output the shared secret derived keying material or an error indicator.
7. If SP 800-56A-compliant key confirmation is required, see section 6.

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

Table 2: One-Pass Diffie-Hellman Key-agreement Scheme Summary

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

[13] This term does not exist in SP 800-56A. SP 800-56A uses Z to denote both quantities: the Z output from the ECC CDH primitive and the byte-string form of Z.

[14] See section 5 for the NIST Concatenation Key Derivation Function. See section 8 and the referenced IETF RFCs for Suite B key derivation functions for the IETF protocols IKE/IPSec, S/MIME, SSH, and TLS.

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

The shared secret Z is computed using the domain parameters D (see Appendix A), the other party's public key and one's own private key. Assume that the party performing the computation is Party A and the other party is Party B. Note that Party A could be either the initiator U or the responder V.

Input:

1. $(q, FR, a, b\{, SEED\}, G, n, h)$: Domain parameters (see appendix A)
2. d_A: One's own private key, and
3. Q_B: The other party's public key.

Process:

1. Compute the point $P = hd_AQ_B$. Note that for the Suite B curves, the cofactor h = 1, which reduces "cofactor ECDH" to "ordinary ECDH".
2. If $P = O$, the point at infinity, output an error indicator.
3. $Z = x_P$ where x_P is the x-coordinate of P.

Output: The shared secret Z or an error indicator.

5. Key Derivation Function – NIST Concatenation KDF

The NIST Concatenation KDF is the required key derivation function unless using one of the IETF protocols for which an exemption has been granted by NIST (see section 8).

The NIST Concatenation KDF is as follows:

Function call: kdf($Z_{bytestring}$[15], *OtherInput*)
where *OtherInput* is *keydatalen* and *OtherInfo*.

Fixed Values (implementation dependent):

1. *hashlen*: an integer that indicates the length (in bits) of the output of the hash function used to derive blocks of secret keying material. This will be either 256 (for SHA-256) or 384 (for SHA-384).
2. *max_hash_inputlen*: an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function.

[15] This term does not exist in SP 800-56A. SP 800-56A uses Z to denote both quantities: the Z output from the ECC CDH primitive and the byte-string form of Z.

Function: H: a hash function, either SHA-256 or SHA-384.

Input:

1. $Z_{bytestring}$: the shared secret in byte-string form.
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 *hashle*n x $(2^{32}-1)$.
3. *OtherInfo*: A bit string equal to the following concatenation:
   AlgorithmID || *PartyUInfo* || *PartyVInfo* {||*SuppPubInfo*} {||*SuppPrivInfo*}
   where the subfields are defined as follows:
   1. *AlgorithmID*: A bit string that indicates how the derived secret keying material will be parsed and for which algorithm(s) the derived secret keying material will be used.
   2. *PartyUInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by Party *U* to the key derivation process. At a minimum, *PartyUInfo* shall include ID_U, the identifier of Party *U*.
   3. *PartyVInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by Party *V* to the key derivation process. At a minimum, *PartyVInfo* shall include ID_V, the identifier of Party *V*.
   4. (Optional) *SuppPubInfo*: A bit string containing additional, mutually-known public information.
   5. (Optional) *SuppPrivInfo*: A bit string containing additional, mutually-known private information (for example, a shared secret symmetric key that has been communicated through a separate channel).

Each of the three subfields *AlgorithmID*, *PartyUInfo* and *PartyVInfo* shall be the concatenation of an application-specific, fixed sequence of substrings of information. Each substring representing a separate unit of information shall have one of these two formats: either it is a fixed-length bit string, or it has the form *Datalen*||*Data*, where *Data* is a variable-length string of zero or more bytes and *Datalen* is a fixed-length, big endian counter that indicates the length (in bytes) of *Data*. (In this variable-length format, a null string of data shall be represented by using *Datalen* to indicate that *Data* has length zero.) An application using this KDF shall specify the ordering and number of separate information substrings used in each of the subfields, *PartyUInfo* and *PartyVInfo*, and shall also specify which of two formats (fixed-length or variable-length) is used for each substring. The application shall specify the lengths for all fixed-length quantities, including the *Datalen* counters.

The subfields *SuppPubInfo* and *SuppPrivInfo* (when allowed by the application) shall be formed by the concatenation of an application-specific, fixed sequence of substrings of additional information that may be used in key derivation upon mutual agreement of parties *U* and *V*. Each substring representing a separate unit of information shall be of the form *Datalen*||*Data*, where *Data* is a variable-length string of zero or more bytes and *Datalen* is a fixed-length, big-endian counter that indicates the length (in bytes) of *Data*.

The information substrings that parties *U* and *V* choose not to contribute are set equal to Null and are represented in this variable-length format by setting *Datalen* equal to zero. If an application allows the use of the *OtherInfo* subfield *SuppPrivInfo* and/or the subfield *SuppPubInfo*, then the application shall specify the ordering and the number of additional information substrings that may be used in the allowed subfield(s) and shall specify the fixed-length of the *Datalen* counters.

Process:

1. $reps = \lceil keydatalen / hashlen \rceil$
2. If $reps > (2^{32}-1)$, then ABORT: output an error indicator and stop.
3. Initialize a 32-bit, big-endian bit string *counter* as 00000001_{16}.
4. If (*counter* || $Z_{bytestring}$ || *OtherInfo*) is more than *max_hash_inputlen* bits long, then ABORT: output an error indicator and stop.
5. For $i = 1$ to *reps* by 1, do the following:
   Compute $Hash_i = H(counter \| Z_{bytestring} \| OtherInfo)$
   Increment *counter* (modulo 2^{32}), treating it as an unsigned 32-bit integer.
6. Let *Hhash* be set to $Hash_{reps}$ if (*keydatalen* / *hashlen*) is an integer; otherwise, let *Hhash* be set to the (*keydatalen* mod *hashlen*) leftmost bits of $Hash_{reps}$.
7. Set $DerivedKeyingMaterial = Hash_1 \| Hash_2 \| \ldots \| Hash_{reps-1} \| Hhash$.

Output:
The bit string *DerivedKeyingMaterial* of length *keydatalen* bits (or an error indicator). 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*. Any call to the key derivation function involving an attempt to hash a bit string that is greater than *max_hash_inputlen* bits long shall cause the KDF to output an error indicator and stop without outputting *DerivedKeyingMaterial*.

Notes:

1. ID_U and ID_V shall be represented in *OtherInfo* as separate units of information, using either the fixed-length format or the variable-length format described above – according to the requirements of the application using this KDF. The rationale for including the identifiers in the KDF input is provided in Appendix B of SP 800-56A.
2. Party *U* shall be the initiator and Party *V* shall be the responder, as assigned by the protocol employing the key-agreement scheme used to determine the shared secret $Z_{bytestring}$.

The output from the KDF shall only be used for secret keying material, such as a symmetric key used for data encryption or message integrity, a secret initialization vector, or a master key that will be used to generate other keys. Non-secret keying material shall not be generated using the shared secret.

Each call to the KDF requires a freshly computed shared secret and this shared secret shall be zeroized immediately following its use. The derived secret keying material shall be computed in it entirety before outputting any portion of it.

The derived secret keying material may be parsed into one or more keys or other secret cryptographic keying material (for example, secret initialization vectors). If Key Confirmation or implementation validation testing are to be performed, then the MAC key shall be formed from the first bits of the KDF output and zeroized after its use.

6. Key Confirmation

Key Confirmation refers to methods used to provide assurance to one party (the key-confirmation recipient) that another party (the key-confirmation provider) possesses the correct shared secret and/or derived keying material (from the key-confirmation recipient's perspective). It is strongly recommended that new developments include key confirmation due to the security benefits that key confirmation provides.

Often, key confirmation is provided implicitly (e.g., by the recipient's ability to successfully decrypt an encrypted message from the other party using a key derived during the key-agreement transaction), but this assurance can also be provided as an integral part of a key-agreement transaction by the explicit exchange of key-confirmation information (e.g., by the exchange of appropriately-defined MAC values computed using a key derived specifically for that purpose).

Under certain circumstances, explicit key confirmation may be incorporated directly into the key-agreement scheme employed by a key-agreement protocol. More often, explicit key confirmation is handled as a separate component of the protocol. The specifications in SP 800-56A deal only with situations where key confirmation is implemented as part of the key-agreement scheme itself, and are limited to cases where the key-confirmation provider has an identifier that is bound to a static public key-establishment key used during the transaction. SP 800-56A does not prohibit the inclusion of key confirmation as a separate component of a protocol (or by any other means), but provides no statement concerning the adequacy of methods that are not specified in SP 800-56A.

The implication of this restriction for Suite B is that SP 800-56A-compliant key confirmation can only be directly incorporated into implementations of the One-Pass Diffie-Hellman scheme, where the responder (as the owner of the static public key-establishment key) serves as the key-confirmation provider and the other party (the initiator of the key-agreement transaction, who contributes an ephemeral public key) serves as the key-confirmation recipient. However, there is no prohibition against providing/obtaining key confirmation by other means and SP 800-56A recommends that each party to a key-agreement transaction should endeavor to obtain assurance that the other party possesses the correct derived keying material.

The NIST SP 800-56A key-confirmation process requires the use of a Message Authentication Code (MAC) algorithm. The Keyed-Hash Message Authentication Code (HMAC), as specified in FIPS 198-1, is to be used with either SHA-256 or SHA-384.

A final note: When it is included in a key-agreement transaction, properly-defined key confirmation may also permit the recipient to obtain assurance that the provider had knowledge of the private key corresponding to the provider's public key and was able to use it correctly. For example, SP 800-56A-compliant key confirmation can be used to provide/obtain assurance of static-private-key possession in conjunction with key-agreement transactions employing the One-Pass Diffie-Hellman scheme.

NIST 800-56A Key-confirmation Steps for One-Pass ECDH:

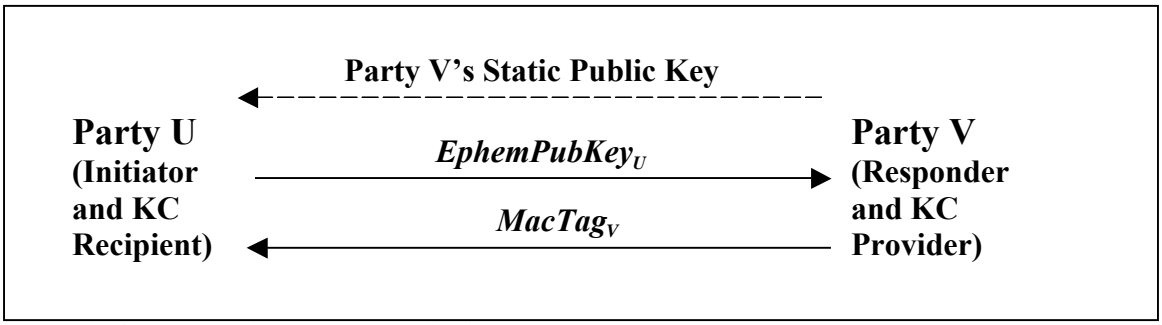

Figure 3: One-Pass ECDH with Key Confirmation

$MacData_V$ is formed as follows:

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

where ID_V is V's identifier, ID_U is U's identifier, *Null* is the empty byte string, $EphemPubKey_U$ is U's ephemeral public key, and *Text* is an optional bit string that may be used during key confirmation and that is known by the parties establishing the secret keying material.

After computing the shared secret and applying the key derivation function to obtain *DerivedKeyingMaterial*, Party V parses *DerivedKeyingMaterial* into two parts, *MacKey* and *KeyData*:

$$MacKey \parallel KeyData = DerivedKeyingMaterial$$

Party V computes

$$MacTag_V = \text{MAC}(MacKey, MacLen, MacData_V),$$

where the mutually-agreed upon parameter *MacLen* determines the length (in bits) of the MAC output.

Party V then sends $MacTag_V$ to Party U.

Party U computes its own versions of $MacData_V$, *MacKey*, *KeyData* and $MacTag_V$ in the same manner as Party V, and then compares its computed $MacTag_V$ to the $MacTag_V$ received from Party V. If the received value is equal to the value computed by Party U, then Party U is assured that Party V has derived the same value for *MacKey* and that Party V shares Party U's value of $MacData_V$. The assurance of a shared value for *MacKey* provides assurance to Party U that Party V also shared the secret values, Z and $Z_{bytestring}$, from which *MacKey* and *KeyData* are derived. Thus, Party U also has assurance that Party V could compute *KeyData* correctly.

7. Assurance of Possession of Static Private Keys

When static key pairs are locally generated, then as part of the certificate request/issuance process, the CA must obtain assurance that the party for whom the certificate is requested (the eventual "owner" of the key pair) is in possession of the private key corresponding to the public key that will be included in the certificate. Methods that can be used by the CA to obtain this assurance of possession are specified in the "Suite B Profile of Certificate Management over CMS."

This requirement applies to the static key-establishment key used in the One-Pass Diffie-Hellman scheme.

8. IETF Protocols

The IETF protocols IKE/IPsec, S/MIME, SSH and TLS (among others) were all developed prior to NIST SP 800-56A and their key-establishment components do not fully comply with the key-establishment requirements contained in SP 800-56A. For example, IETF protocols often permit the use of domain parameters, hash functions, MAC algorithms, etc., that would not be approved by SP 800-56A and are not recommended for use in Suite B products.

In addition, there are discrepancies in the methods used to derive keying material from the shared secret value created during what could otherwise be a NIST-compliant ECDH exchange. NIST has, however, made exceptions for such discrepancies between SP 800-56A and a number of these widely-used protocols. See section 7.1 of the "Implementation guidance for FIPS 140-2 and the Cryptographic Module Validation Program (CMVP)" for a description of the extent to which departures from the requirements of SP 800-56A are allowed in products submitted to the CMVP for FIPS 140-2 validation.

The public key infrastructure certificates used in each protocol follow the "Suite B Certificate and Certificate Revocation List Profile." This profile is a refinement of RFC 5280, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," and assumes that the reader is familiar with RFC 5280.

Certificate management follows the "Suite B Profile of Certificate Management over CMS[16]."

The following sections discuss implementing specific IETF protocols in the context of Suite B.

8.1 IKE/IPsec

Suite B compliant implementations of IPsec must use one of the four cryptographic suites defined in RFC 4869[17], "Suite B Cryptographic Suites for IPsec," by L. Law & J. Solinas, dated May 2007. The IPsec key-agreement protocols, IKEv1 (see RFC 2409) and IKEv2 (see RFC 4306), use the Ephemeral Unified Model key-agreement scheme.

In Suite B compliant IKEv2, the key exchange must be authenticated (by both parties) using ECDSA signatures. Therefore, each party must possess an X.509v3 certificate containing an ECDSA public key (signed by a CA with ECDSA).

Similarly, in Suite B compliant IKEv1, the key exchange must be authenticated (by both parties). Authentication can be accomplished using ECDSA signatures, in which case, each party must possess an X.509v3 certificate containing a cryptographic-suite-dependent ECDSA public key (signed by a CA with ECDSA). For interoperability purposes, Suite B compliant IKEv1 implementations must also support authentication via the use of a pre-shared key.

IKEv1 and IKEv2 do not use the NIST Concatenation KDF. Section 5 of RFC 2409 and section 2.13 of RFC 4306 describe the PRF used to derive session keys for encryption and integrity protection. The PRF used for key derivation by Suite B IPsec implementations employs HMAC based on either SHA-256 or SHA-384, and has the structure of the NIST Concatenation KDF in feedback mode (with an iteration variable described in section 5.2 of SP 800-108).

The NIST key-confirmation routine is only intended for protocols that use static key pairs. Suite B implementations of IKE will not use static key pairs, hence NIST key confirmation cannot be included in Suite B implementations of IKE. However, the authentication processes used in IKE include implicit confirmation that the keying material has been correctly computed.

8.2 S/MIME

Suite B compliant implementations of S/MIME must follow RFC 5008 "Suite B in Secure/Multipurpose Internet Mail Extensions (S/MIME)", by R. Housley & J. Solinas, dated September 2007.

Suite B compliant S/MIME utilizes the One-Pass ECDH key-agreement scheme (using either the P-256 or P-384 elliptic curve) to facilitate e-mail encryption. The initiator's

[16] Cryptographic Message Syntax

[17] RFC 4869 is in the process of being updated. The update will appear as an Internet-Draft on the IETF website (www.ietf.org) and once approved will become a new RFC .

contribution is an ephemeral key, while the recipient's contribution is static. Therefore, in order to receive encrypted e-mail, the recipient must possess an X.509v3 certificate containing the static public ECDH key-establishment key (signed by a CA using ECDSA) and that certificate must be provided to the initiator. The initiator generates an ephemeral ECDH key on the same curve as the recipient's static public ECDH key.

The initiator must have an ECDSA signing key in order to send a signed message to the recipient. The recipient must be provided an X.509v3 certificate containing the public ECDSA key that will be used to verify the initiator's signature. (That certificate must be signed by a CA that also used ECDSA.)

Suite B compliant implementations of S/MIME are not permitted to use certificates asserting both key agreement/encipherment and digital signature in the key usage extension.

S/MIME does not use the NIST Concatenation KDF. The KDF used in S/MIME for elliptic curve cryptography is taken from "SEC1: Elliptic Curve Cryptography", section 3.6.1 and is the ANSI X9.63 KDF. This KDF is similar to the NIST Concatenation KDF except that the order of the counter and the shared secret are reversed and some of the requirements on the other input values are relaxed. For Suite B, the only hash algorithms that may be used are SHA-256 and SHA-384 (see RFC 5008).

8.3 SSH

Suite B compliant implementations of SSH must follow the Internet-Draft "Suite B Cryptographic Suites for Secure Shell" (draft-igoe-secsh-suiteb-00.txt), by K. Igoe, dated September 2008. Note that the September 2008 draft is a work-in-progress. Subsequent drafts and eventually the final RFC will supersede earlier drafts.

In Suite B compliant SSH, the key-agreement protocol employs the Ephemeral Unified Model ECDH key-agreement scheme (using either the P-256 or P-384 elliptic curve), in which the server authenticates its contribution via an ECDSA signature. The server must supply to the client an X.509v3 certificate containing the public ECDSA key that will be used to verify its signature. (That certificate must be signed by a CA that also used ECDSA.)

While the client does not directly authenticate its contribution to the key-agreement process, the client does authenticate itself to the server by sending an ECDSA-signed message via a secure tunnel that is protected by keying material established during the key-agreement protocol. The SSH authentication protocol refers to this as the "public key" method of client authentication. The "public key blob" of the client's ECDSA-signed SSH_MSG_USERAUTH_REQUEST message must include an X.509v3 certificate (signed by a CA with ECDSA) that contains the public ECDSA signature key that will be used to verify the client's signature.

Note that neither the server nor the client in a Suite B compliant SSH implementation requires a static key-establishment key pair.

SSH does not use the NIST Concatenation KDF. Section 7.2 of RFC 4253 describes the KDF used to compute IVs, encryption keys and integrity keys. The SSH KDF differs from the NIST Concatenation KDF in that SSH uses feedback instead of a counter and the SSH KDF lacks some of the parameters used by the NIST Concatenation KDF. For Suite B, the hash function used for the SSH KDF will be either SHA-256 or SHA-384.

Section 4 of "Elliptic-Curve Algorithm Integration in the Secure Shell Transport Layer" (draft-green-secsh-ecc-08.txt, work-in-progress) requires that all elliptic-curve public keys must be validated after they are received. See appendix B.3 of the Suite B Implementer's Guide to SP 800-56A for the public-key validation routine.

The NIST key-confirmation routine is only intended for parties that make use of a static key-establishment key pair. Suite B implementations of SSH will not use static key pairs, hence NIST key confirmation cannot be included in Suite B implementations of SSH. Implicit key confirmation is provided by the client by sending SSH_MSG_USERAUTH_REQEST protected using established keys.

8.4 TLS

Suite B implementations of TLS follow: RFC 5246 "The Transport Layer Security (TLS) Protocol Version 1.2" and RFC 5430 "Suite B Profile for Transport Layer Security (TLS)".

During the TLS handshake protocol, Suite B compliant implementations supporting a common level of security (as defined in RFC 5430) will utilize the form of key-agreement that TLS calls ECDHE_ECDSA (using either the P-256 or P-384 elliptic curve, as appropriate to the security level). This is the Ephemeral Unified Model key-agreement scheme, in which the server authenticates its contribution via an ECDSA signature. The server must supply to the client an X.509v3 certificate containing an acceptable (to the client) ECDSA key that will be used to verify its signature. That certificate must be signed by a CA using ECDSA.

If the client is also to be authenticated during the TLS handshake, it must possess an X.509v3 certificate containing an acceptable (to the server) ECDSA key, signed by a CA using an acceptable version of ECDSA. To indicate its desire for client authentication, the server sends a certificate request message of the type "ECDSA_sign," asking the client for an appropriate ECDSA certificate.

RFC 5430 defines which types of ECDSA keys and certificate signatures are acceptable for Suite B compliance with the defined security levels. Note that neither the server nor the client in a TLS handshake requires a static key-establishment key pair to establish a Suite B compliant connection.

TLS does not use the NIST Concatenation KDF. Section 5 of RFC 5246 defines one PRF used for key derivation based upon HMAC which is acceptable for Suite B use, provided

that the hash function used by HMAC is SHA-256 or SHA-384. The hash function used will depend on the target security level.

The NIST key-confirmation routine is only intended for parties that make use of a static key-establishment key pair. Suite B implementations of TLS (supporting a common level of security) will use only ephemeral key-establishment keys; hence, there is no requirement for the NIST key-confirmation routine. Note that a form of explicit key confirmation is provided by the exchange of "FINISHED" messages at the conclusion of the handshake protocol.

References

ANS X9.63, "Public Key Cryptography for the Financial Services Industry: Key-Agreement and Key Transport Using Elliptic Curve Cryptography", dated December 2001.

FIPS 180-3, Secure Hash Standard, Issued October 2008.

FIPS 186-3, Digital Signature Standard, Issued June 2009.

FIPS 198-1, The Keyed-Hash Message Authentication Code, Issued July 2008.

"Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program", published by NIST, initial release March 2003/update April 2009.

Internet-Draft "Elliptic-Curve Algorithm Integration in the Secure Shell Transport Layer", by D. Stebila and J. Green, dated June 2009 (draft-green-secsh-ecc-08.txt), work-in-progress.

Internet-Draft "Suite B Cryptographic Suites for Secure Shell", by K. Igoe, dated September 2008 (draft-igoe-secsh-suiteb-00.txt), work-in-progress.

Internet-Draft "Suite B Certificate and Certificate Revocation List (CRL) Profile", by J. Solinas and L. Zieglar, dated July 2009 (draft-solinas-suiteb-cert-profile-04.txt), work-in-progress.

Internet-Draft "Suite B Profile of Certificate Management over CMS", by S. Turner and M. Peck, dated April 2009 (draft-turner-suiteb-cmc-00.txt), work-in-progress.

"Mathematical Routines for NIST Prime Elliptic Curves," dated March 2008, available on the Information Assurance Suite B Cryptography webpage at www.nsa.gov.

RFC 4253 "The Secure Shell (SSH) Transport Layer Protocol", by T. Ylonen and C. Lonvick, dated January 2006.

RFC 4306 "Internet Key Exchange (IKEv2) Protocol, by C. Kaufman, dated December 2005.

RFC 4869 "Suite B Cryptographic Suites for IPsec", by L. Law & J. Solinas, dated May 2007. (Note: This RFC is being updated. Check the IETF website: www.ietf.org)

RFC 5008 "Suite B in Secure/Multipurpose Internet Mail Extensions (S/MIME)", by R. Housley & J. Solinas, dated September 2007.

RFC 5246 "The Transport Layer Security (TLS) Protocol Version 1.2", by T. Dierles and E. Rescorla, dated August 2008.

RFC 5430 “Suite B Profile for Transport Layer Security (TLS)”, by R. Housley, E. Rescorla and M. Salter, dated March 2009.

Standards for Efficient Cryptography, “SEC1: Elliptic Curve Cryptography”, dated September 2000.

SP 800-56A “NIST SP 800-56A: Recommendation for Pair-Wise Key-establishment Schemes Using Discrete Logarithm Cryptography”, by E. Barker, D. Johnson and M. Smid, dated March 2007.

SP 800-57 “NIST SP 800-57: Recommendation for Key Management – Part 1: General”, by E. Barker, W. Barker, W. Burr, W. Polk and M. Smid, dated March 2007.

SP 800-108 “NIST SP 800-108: Recommendation for Key Derivation Using Pseudorandom Functions, by Lily Chen, dated November 2008.

Appendix

A. Suite B Curves and Domain Parameters

Domain parameters for ECC schemes are of the form: (q, FR, a, b{, $SEED$}, G, n, h), where q is the field size; FR is an indication of the basis used; a and b are two field elements that define the equation of the curve; $SEED$ is an optional bit string that is included if the elliptic curve was randomly generated in a verifiable fashion; G is a generating point consisting of (x_G,y_G) of prime order on the curve; n is the order of the point G; and h is the cofactor (which is equal to the order of the curve divided by n).

Suite B requires the use of one of the following two sets of domain parameters:

A.1. Domain Parameters for Curve P-256

The values for the domain parameters for P-256 follow:

Field size:
q = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFF

Field Representation indicator
FR = NULL

Curve parameter:
a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFC

Curve parameter:
b = 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6
3BCE3C3E 27D2604B

Seed used to generate parameter b:
SEED = C49D3608 86E70493 6A6678E1 139D26B7 819F7E90

x-coordinate of base point G:
x_G = 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0
F4A13945 D898C296

y-coordinate of base point G:
y_G = 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE
CBB64068 37BF51F5

Order of the point G:
n = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84
F3B9CAC2 FC632551

Cofactor (order of the elliptic curve divided by the order of the point G)
h = 1

A.2 Domain Parameters for Curve P-384

The values for the domain parameters for P-384 follow:

Field size:
```
q =  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
     FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFF
```

Field Representation indicator
FR = NULL

Curve parameter:
```
a =  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
     FFFFFFFF FFFFFFFE FFFFFFFF 00000000 00000000 FFFFFFFC
```

Curve parameter:
```
b =  B3312FA7 E23EE7E4 988E056B E3F82D19 181D9C6E FE814112
     0314088F 5013875A C656398D 8A2ED19D 2A85C8ED D3EC2AEF
```

Seed used to generate parameter b:
```
SEED = A335926A A319A27A 1D00896A 6773A482 7ACDAC73
```

x-coordinate of base point G:
```
xG =  AA87CA22 BE8B0537 8EB1C71E F320AD74 6E1D3B62 8BA79B98
      59F741E0 82542A38 5502F25D BF55296C 3A545E38 72760AB7
```

y-coordinate of base point G:
```
yG = 3617DE4A 96262C6F 5D9E98BF 9292DC29 F8F41DBD 289A147C
     E9DA3113 B5F0B8C0 0A60B1CE 1D7E819D 7A431D7C 90EA0E5F
```

Order of the point G:
```
n =  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
     C7634D81 F4372DDF 581A0DB2 48B0A77A ECEC196A CCC52973
```

Cofactor (order of the elliptic curve divided by the order of the point G)
h = 1

B. Key Pair Generation and Assurance of Public Key Validity

Each static and ephemeral private key *d* and public key *Q* shall be generated using the appropriate domain parameters and either "Key Pair Generation Using Extra Random Bits" or "Key Pair Generation by Testing Candidates." Both generation methods are

contained in FIPS 186-3, Appendix B.4 on ECC Key Pair Generation and detailed in appendices B.1 and B.2 of this document for convenience.

Both methods require use of an approved random bit generator, referred to as an RBG in the description of each method.

The process from NIST SP 800-56A to provide assurance of public key validity is given in section B.3. Note that both ECDH schemes use this process. Assurance of public key validity is inherent in both key generation methods; hence, additional steps to provide assurance of public-key validation immediately after generation are not required.

B.1 Key Pair Generation Using Extra Random Bits

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

The steps are as follows:

Input: Domain Parameters (*q*, *FR*, *a*, *b*{, *SEED*}, *G*, *n, h*)

Output:

1. status: The status returned from the key pair generation procedure. The status will indicate SUCCESS or an ERROR.
2. (*d*, *Q*): The generated private and public keys. If an error is encountered during the generation process, invalid values for *d* and *Q* should be returned, as represented by *Invalid_d* and *Invalid_Q*. *d* is an integer and *Q* is an elliptic curve point. The generated private key *d* is in the range [1, n-1].

Process:

1. N = len(n). Check that N is valid, that is, N = 256 or N = 384 (the only valid lengths for Suite B)
2. If N is invalid, then return an ERROR indication, *Invalid_d* and *Invalid_Q*.
3. *requested_security_strength* = the security strength associated with N(either 128 when using P-256 or 192 when using P-384).
4. Obtain a string of N+64 *returned_bits* from an RBG with a security strength of *requested_security_strength* or more. If an ERROR indication is returned, then return the ERROR indication, *Invalid_d* and *Invalid_Q*.
5. Convert returned_bits to the (non-negative) integer *c* (see appendix C.1).
6. $d = (c \bmod (n-1)) + 1$
7. $Q = dG$.
8. Return SUCCESS, *d*, and *Q*.

B.2 Key Pair Generation by Testing Candidates

In this method, a random number is obtained and tested to determine that it will produce a value of *d* in the correct range. If *d* is out-of-range, the process is iterated until an acceptable value of *d* is obtained.

The steps are as follows:

Input: Domain Parameters (*q*, *FR*, *a*, *b*{, *SEED*}, *G*, *n, h*)

Output:

1. status: The status returned from the key pair generation procedure. The status will indicate SUCCESS or an ERROR.
2. (*d*, *Q*): The generated private and public keys. If an error is encountered during the generation process, invalid values for *d* and *Q* should be returned, as represented by *Invalid_d* and *Invalid_Q*. *d* is an integer and *Q* is an elliptic curve point. The generated private key *d* is in the range [1, n-1].

Process:

1. N = len(n). Check that N is valid, that is, N = 256 or N = 384 (the only valid lengths for Suite B)
2. If N is invalid, then return an ERROR indication, *Invalid_d* and *Invalid_Q*.
3. *requested_security_strength* = the security strength associated with N (either 128 when using P-256 or 192 when using P-384).
4. Obtain a string of N *returned_bits* from an RBG with a security strength of *requested_security_strength* or more. If an ERROR indication is returned, then return the ERROR indication, *Invalid_d* and *Invalid_Q*.
5. Convert *returned_bits* to the (non-negative) integer *c* (see appendix C.1).
6. If (*c* > n-2), then go to step 4.
7. *d* = *c* + 1
8. *Q* = *dG*.
9. Return SUCCESS, *d*, and *Q*.

B.3 Assurance of Public Key Validity

Prior to executing a routine that performs elliptic curve computations on either ephemeral or static public keys, the ECC Partial Public-Key Validation Routine should be executed on the public keys.

SP 800-56A specifies two routines to perform public-key validation: ECC Full Public Key Validation and ECC Partial Public Key Validation. The difference between the two

routines is a check to ensure that the point has the correct order. This check is unnecessary for prime-order curves, such as the curves used in Suite B. As long as the implementation under testing only claims to support the Suite B subset of NIST curves, the partial validation routine will be sufficient to satisfy FIPS 140 CAVP testing of both full and partial public-key validation capabilities.

ECC Partial Public-Key Validation Routine

Input: The appropriate domain parameters and $Q = (x_Q, y_Q)$, a candidate ECC public key.

Process

1. Verify that Q is not the point at infinity O. This can be done by inspection if the point is entered in the standard affine representation. If the point can be represented in standard affine representation, then the point is not the point at infinity.
2. Verify that x_Q and y_Q are integers in the interval $[0,q-1]$. This ensures that each coordinate of the public key has the unique correct representation of an element in the underlying field.
3. Verify that $(y_Q)^2 \equiv (x_Q)^3 + ax_Q + b \pmod{q}$, where a, b, q are the values indicated in the domain parameters used (see appendix A). This ensures that the public key is on the correct elliptic curve.

Note that in SP 800-56A, the interval in step 2 is specified as $[0,p-1]$ and the modulus in step 3 is specified as p. For the Suite B curves, $q = p$.

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

C. Data Conversions

C.1 Conversion of a Bit String to an Integer

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

$$\{x_1, \ldots, x_m\} \rightarrow (x_1 * 2^{m-1}) + (x_2 * 2^{m-2}) + \ldots + (x_{m-1} * 2) + x_m.$$

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

Input: The bit string $b_1, b_2, \ldots, b_m$ to be converted.

Output: The requested integer representation C of the bit string.

Process:

1. Let $(b_1, b_2, \ldots, b_m)$ be the bits of b from leftmost to rightmost.

2. $C = \sum 2^{(m-i)} b_i$ for $i = 1$ to m.
3. Return C

C.2 Conversion of an Integer to a Bit String

An integer x in the range $0 \leq x < 2^m$ may be converted to an m-long sequence of bits by using its binary expansion as shown below:

$$(x_1 * 2^{m-1}) + (x_2 * 2^{m-2}) + \ldots + (x_{m-1} * 2) + x_m \rightarrow \{x_1, \ldots, x_m\}.$$

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

Input: A non-negative integer C to be converted and the intended length m of the bit string satisfying $C < 2^m$.

Output: The representation $b_1, b_2, \ldots, b_m$ of the integer C.

Process:

1. Input C and m.
2. For a given integer m that satisfies $C < 2^m$, the bits b_i shall satisfy:

   $C = \sum 2^{(m-i)} b_i$ for $i = 1$ to m.
3. Return $(b_1, b_2, \ldots, b_m)$.

C.3 Field-Element-to-Byte String/Integer-to-Byte String Conversion

This section combines the Field-Element-to-Byte String and Integer-to-Byte String conversions from SP 800-56A because for the Suite B curves, the field elements are integers in the interval [0,p-1].

Input: A non-negative integer C and the intended length m of the byte string satisfying

$$2^{8m} > C$$

Output: A byte string S of length m bytes.

Process:

1. Let $S_1, S_2, \ldots, S_m$ be the bytes of S from leftmost to rightmost.
2. The bytes of S shall satisfy:

   $C = \sum 2^{8(m-i)} S_i$ for $i = 1$ to m,

   where S_i is interpreted as the big-endian binary representation of an unsigned integer.