HASH	A cryptographic hash function, such as SHA-1.
HMAC-HASH	The HMAC algorithm using the hash function, HASH (e.g., HASH could be SHA-1). See FIPS 198-1 for the specification of the HMAC algorithm using one of the **approved** hash functions.
HMAC-PRF	The HMAC function being used as a PRF.
P_HASH	A function that uses the HMAC-HASH as the core function in its construction. The specification of this function is in RFCs 2246 and 5246.
a | x	a divides x.
$\lceil x \rceil$	The ceiling of x; the smallest integer $\geq x$. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.2 \rceil = 6$.
$\lfloor x \rfloor$	The floor of x; the largest integer $\leq x$. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 6.9 \rfloor = 6$.
len(x)	The length of the string x in bits.
$\oplus$	A bitwise logical operation such that $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$, and $0 \oplus 1 = 1$. For example, given a string $A = 10$ and a string $B = 11$, then $A \oplus B = (1 \oplus 1) \| (0 \oplus 1) = 01$.
XOR	A *XOR* B is equivalent to A $\oplus$ *B*. See the definition of the bitwise logical operation $\oplus$ above.
Pre-shared-key	A secret key that has previously been established. See "pre-shared key" in Section 3.1 above.
SKEY	A session key in TPM.

4 Extraction-then-Expansion (E-E) Key Derivation Procedure

NIST has specified several key derivation functions (KDFs) in SP 800-56A, SP 800-56B and SP 800-108 [SP 800-108]. SP 800-56C [SP 800-56C] specifies an additional KDF that is an extraction-then-expansion (E-E) procedure, which has a different structure from the KDFs in SP 800-56A, SP 800-56B and SP 800-108. The procedure consists of two separate steps: a randomness extraction step and a key expansion step. The general specification of the E-E procedure is in SP 800-56C, which provides an additional method for deriving keys when performing a key establishment scheme as

specified in SP 800-56A and SP 800-56B . Figure 1 below shows the relationship of the E-E procedure in SP 800-56C with the **approved** KDFs in SP 800-108. In Figure 1, the randomness extraction step outputs a key derivation key, which is then used as input to the key expansion step. Any **approved** KDFs in SP 800-108 can be used as the key expansion step that derives keying material and can also be used to derive more keying material from the key expansion step's derived key(s)/keying material. The following sections discuss protocols that use the E-E procedure. Note that even though the KDFs in these protocols use the E-E procedure for key derivation, their specific randomness extraction and/or key expansion step(s) do not meet all specification of SP 800-56C. These KDFs are only **approved** for use within the limitations described in the corresponding sections later in this document.

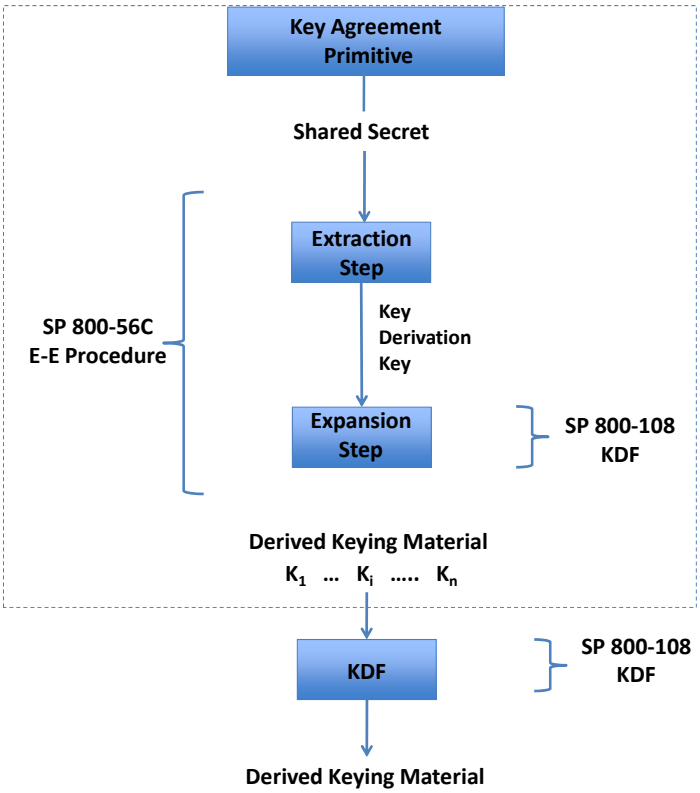


Figure 1: Key Derivation

4.1 Internet Key Exchange (IKE)

Versions 1 and 2 of the Internet Key Exchange (IKE) are specified in RFC 2409 and RFC 4306, respectively, and use HMAC-based Pseudorandom Functions (PRFs) in

their KDFs to generate keying material for their security associations (SAs). IKEv1[1] and IKEv2 specify multiple PRFs, including those based on HMAC and block ciphers. Only an **approved** hash function and HMAC function **shall** be used.

4.1.1 IKE version 1 (IKEv1)

In IKEv1, a string called *SKEYID* is derived from secret material known only to the communicating parties. An HMAC-PRF is used to produce the *SKEYID*. The secret material that is input to the HMAC function is either a Diffie-Hellman shared secret g^{xy}, secret nonces generated by both of the communicating parties, or a pre-shared key. In the protocol, the method used to generate *SKEYID* depends on the authentication method. One of the following three different functions is used as a randomness extraction step to produce the *SKEYID*:

1) When digital signatures are used for authentication, the function is:

   SKEYID = HMAC (*Ni_b* || *Nr_b*, g^{xy}), where *Ni_b* and *Nr_b* are non-secret values.

2) When a public key algorithm encryption is used for authentication, the function is:

   SKEYID = HMAC (HASH (*Ni_b* || *Nr_b*), CKY-I || CKY-R), where *Ni_b* and *Nr_b* are secret nonces, and CKY-I and CKY-R are non-secret values.

3) When a pre-shared key is used for authentication, the function is:

   SKEYID = HMAC (*pre-shared-key*, *Ni_b* || *Nr_b*), where *Ni_b* and *Nr_b* are non-secret values.

Additional technical details for the variables in the functions are provided in RFC 2409.

The appropriate PRF (i.e., the HMAC functions 1-3 above) is executed only once to generate the *SKEYID*.

After the randomness extraction step above, the *SKEYID* is fed into a feedback function that performs the key expansion step and produces the necessary keying material. The resulting keying material is defined by the following equations:

SKEYID_d = *HMAC* (*SKEYID,* g^{xy} || *CKY-I* || *CKY-R* || *0*)

(*SKEYID_d* is used as the key derivation key to generate fresh keying material for new, negotiated security associations.)

SKEYID_a = *HMAC* (*SKEYID, SKEYID_d* || g^{xy} || *CKY-I* || *CKY-R* || *1*)

(*SKEYID_a* is used as an HMAC key to authenticate the current security association's messages.)

SKEYID_e = *HMAC* (*SKEYID, SKEYID_a* || g^{xy} || *CKY-I* || *CKY-R* || 2)

[1] Note that IKEv1 is considered obsolete by the IETF and not recommended for new protocols.

(*SKEYID_e* is used as a key derivation key to derive a symmetric encryption key to provide confidentiality for the current SA's messages. More details about this function can be found in Appendix B of RFC 2409.)

CKY-I and *CKY-R* are non-secret values. The technical details of these variables are provided in RFC 2409. Figure 2 below shows the feedback function that produces the *SKEYID_d, SKEYID_a* and *SKEYID_e* above. This function conforms to the specification of the key expansion step in SP 800-56C.

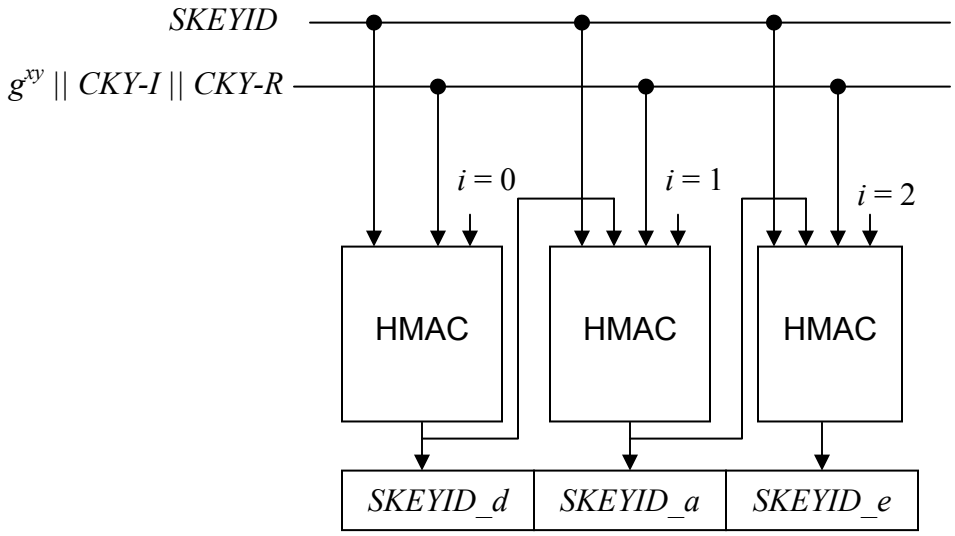


Figure 2: Key expansion Step in IKEv1

Note that both *SKEYID* and the Diffie-Hellman shared secret g^{xy} are secret values and are used as inputs to the HMAC in the key expansion step. Also note that all three resulting keys (i.e., *SKEYID_d*, *SKEYID_a* and *SKEYID_e*) are pseudorandom keys.

The IKEv1[2] KDFs are **approved** when the following conditions are satisfied:

(1) The IKEv1 KDFs are performed in the context of the IKEv1 protocol.

(2) The PRF is an HMAC-based PRF.

(3) The HMAC and HASH are NIST-**approved** algorithms and are used as specified in FIPSs 198-1 [FIPS 198-1] and 180-3, respectively.

[2] Note that when the function (randomness extraction step) used to produce the *SKEYID* is *SKEYID* = HMAC (*Ni_b* || *Nr_b*, g^{xy}) as described earlier in this section, the IKEv1 KDF is compliant with the current specification of SP 800-56C.

4.1.2 IKE version 2 (IKEv2)

In IKEv2, an HMAC is used as a randomness extraction step to extract randomness from a Diffie-Hellman shared secret (g^{ir}) and to uniformly distribute the randomness across the output (*SKEYSEED***)**. The function to produce *SKEYSEED* is:

$$SKEYSEED = \text{HMAC}(Ni \,||\, Nr, g^{ir}).$$

Ni and *Nr* are nonces generated by the protocol initiator and responder, respectively; see RFC 4306 for more details. This function acts as the randomness extraction step of the E-E KDF. After the *SKEYSEED* is generated, a key expansion step is used to derive keys for the security association (SA). *SKEYSEED* is the key derivation key for the key expansion step. The string of concatenated non-secret attributes: *Ni* || *Nr* || *SPIi* || *SPIr*[3] is the fixed input message *P* field in SP 800-56C. Information about these non-secret attributes can be found in RFC 4306. The specification of the key expansion step can be found in Sections 2.13 and 2.14 of the RFC. Note that as shown in Figure 1, the keying material generated by this key expansion step is the output of the E-E procedure.

One of the seven secret keys derived from the *SKEYSEED* is called *SK_d***.** It is used as the key derivation key to derive new keys for child SA(s) using another KDF that is functionally the same as the key expansion step, but with a different set of attributes.

One of the attributes is an optional new Diffie-Hellman shared secret established during the creation of the child SA. Details of this function can be found in Sections 2.17 and 2.18 of RFC 4306.

The IKEv2 KDFs, which are compliant with SP 800-56C, are **approved** when used with an **approved** HMAC function using an **approved** hash function; see FIPSs 198-1 and 180-3, respectively**.** These KDFs are included here for completeness.

4.2 Key Derivation in Transport Layer Security (TLS)

In TLS, after a cipher suite negotiation is completed, a Diffie-Hellman (DH) key agreement or RSA key transport scheme is used to generate a pre-master secret. When RSA key transport is used, the pre-master secret is a random value generated by the client. When the DH key agreement scheme is used, the pre-master secret is the shared secret generated by the key agreement scheme.

4.2.1 Key Derivation in TLS versions 1.0 and 1.1

In TLS versions 1.0 and 1.1 (TLS 1.0 and 1.1), the pre-master secret is input into an HMAC-MD5/HMAC-SHA-1 PRF[4] with some non-secret values to produce a master secret; the PRF acts as the randomness extraction step in an E-E KDF. The master

---

[3] *Ni* and *Nr* are nonces created by the initiator and responder, respectively. *SPIi* and *SPIr* are security parameter indexes (SPIs) of the initiator and responder respectively.

[4] The HMAC-MD5/HMAC-SHA-1 PRF uses HMAC-MD5 and HMAC-SHA-1 as the core functions in its construction, as specified in RFC 2246, Section 5.

secret is then input into the HMAC-MD5/HMAC-SHA1 PRF with other non-secret values to derive keying material for the negotiated cryptographic functions; in this case, the PRF acts as the key expansion step in the E-E KDF.

The HMAC-MD5/HMAC-SHA1 PRF contains two functions: P_MD5 and P_SHA1, which use MD5-HMAC and SHA-1-HMAC as the core functions, respectively. The specifications of P_MD5 and P_SHA-1 are in Section 5 of RFC 2246 (the function called P_hash in the RFC). The P_HASH function (P_MD5 or P_SHA-1) uses the double-pipeline iteration mode and HMAC-PRF specified in SP 800-108.

The outputs from both P_MD5 and P_SHA-1 are *XOR*ed together to produce the PRF output. This PRF is used as both a randomness extraction step to generate the master secret and as a key expansion step to derive keying material for the protocol from the master secret.

The TLS 1.0 and 1.1 KDF is **approved** when the following conditions are satisfied:

(1) The TLS 1.0 and 1.1 KDF is performed in the context of the TLS protocol.

(2) SHA-1 and HMAC are as specified in FIPSs 180-3 and 198-1, respectively.

Note that MD5 and HMAC-MD5 **shall not** be used as a general hash function or HMAC function, respectively.

4.2.2 Key Derivation in TLS version 1.2

In TLS version 1.2 (TLS 1.2), the pre-master secret is input into an HMAC-SHA-256 PRF[5] with some non-secret values to produce a master secret; the PRF acts as the randomness extraction step in an E-E KDF. The master secret is then input into the HMAC-SHA-256 PRF with some other non-secret values to derive keying material for the negotiated cryptographic functions; in this case, the PRF acts as the key expansion step in the E-E KDF.

The HMAC-SHA-256 PRF is P_SHA256. This PRF is used instead of the PRF in TLS 1.0 and 1.1 which is (P_MD5 $\oplus$ P_SHA-1).

In TLS 1.2, in addition to P_SHA256, any P_HASH with a stronger hash function, such as SHA-384 or SHA-512 (in FIPS 180-3), can be used as the PRF.

The TLS 1.2 KDF is an **approved** KDF when the following conditions are satisfied:

(1) The TLS 1.2 KDF is performed in the context of the TLS protocol.

(2) HMAC is as specified in FIPS 198-1.

(3) P_HASH uses either SHA-256, SHA-384 or SHA-512.

[5] The HMAC-SHA-256 PRF uses HMAC-SHA-256 as the core function in its construction, as specified in RFC 5246, Section 5.

5 Other Existing Key Derivation Functions

In this section, several application-specific KDFs that are not specified in SP 800-56A, SP 800-56B, SP 800-56C or SP 800-108, are considered. They do not follow the extraction-then-expansion procedure in SP 800-56C or may not meet all the specification requirements in either SP 800-56A, SP 800-56B or SP 800-108. The following figure illustrates the application of these KDFs.

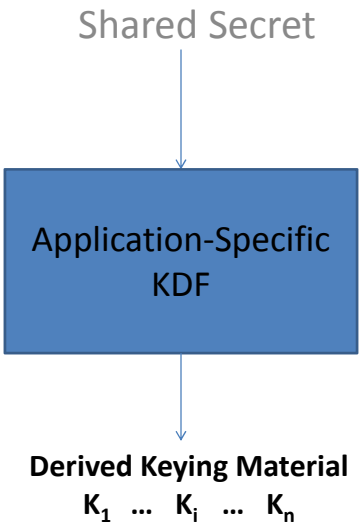


Figure 3: Application-Specific KDF.

5.1 Key Derivation Functions in American National Standards (ANS) X9.42-2001 and ANS X9.63-2001

There are two hash-based KDFs specified in ANS X9.42-2001 and one hash-based KDF specified in ANS X9.63-2001. These hash-based KDF specifications are similar to (but not the same as) the specifications of the hash-based KDFs specified in SPs 800-56A and B. A shared secret that is generated during a key agreement scheme and other non-secret values are used as input to the hash-based KDF to produce the derived keying material for the application. A counter value is pre-pended to the shared secret in the input to the hash-based KDFs in SPs 800-56A and B, but it is appended to the shared secret in the input to the hash-based KDFs in ANS X9.42-2001 and ANS X9.63-2001. Also, the identifiers of the communicating parties are required in the input to the hash-based KDFs in SPs 800-56A and B, but optional in ASN X9.42-2001 and ASN X9.63-2001.

The hash-based KDFs in ANS X9.42-2001 and ANS X9.63-2001 are **approved** when the following conditions are satisfied:

(1) Each of the hash-based KDFs is performed in the context of an ANS X9.42-2001 or ANS X9.63-2001 key agreement scheme.

(2) The hash function is one of the hash functions specified in FIPS 180-3.

(3) The hash function deployed in the hash-based KDF meets the security strength(s) required by the cryptographic function(s) for which the keying material is being generated. The security strengths of **approved** hash functions used in KDFs can be found in SP 800-57 [SP 800-57].

Note that any KDF that meets the specification of either the ANS X9.42-2001 or ANS X9.63-2001 hash-based KDF and is used in a scheme specified in one of these two Standards is **approved** when conditions (2) and (3) above are met. For example, the KDF specified in Section 2.1.2 (Generation of Keying Material) of RFC 2631 and the KDF (specified in [SEC1]) used in RFC 3278 are **approved**.

5.2 Secure Shell (SSH) Key Derivation Function

SSH is a protocol used between clients and servers for secure remote login and other secure network services over an insecure network or the Internet. The Internet Engineering Task Force (IETF) governs the SSH protocol (see RFC 4251). The SSH protocol consists of three major components: the Transport Layer Protocol (RFC 4253 [RFC 4253]), the User Authentication Protocol (RFC 4252 [RFC 4252]) and the Connection Protocol (RFC 4254 [RFC 4254]). The Transport Layer Protocol provides server authentication, confidentiality, and integrity.

Output from a key exchange[6] (see Section 7.2 of RFC 4253) in the Transport Layer Protocol is a shared secret, "*K*", and a hash value, "*H*". In the protocol, *K*, *H* and a specific set of other non-secret values are input to a hash function-based KDF to derive keying material. Different KDFs produce keying material for different cryptographic functions; however, the KDFs are similar (see below). The specifications for the KDFs in the Transport Layer Protocol can be found in Section 7.2 of RFC 4253.

The following is a description of the KDF used to derive a key, where *HASH* denotes a hash function, such as SHA-1 as specified in FIPS 180-3. The key to be derived is denoted by *KEY***,** and the length of *KEY* is denoted by *L*. The length of the hash function output is denoted by *HASH_Length*.

$N = \lceil (L/Hash\_Length) \rceil$

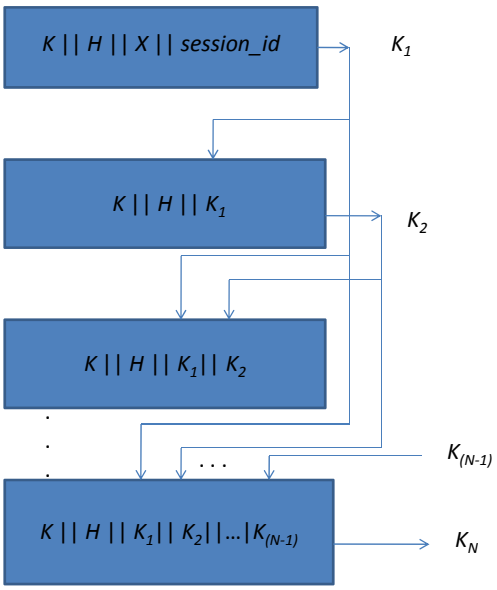
X is a character, such as A, B, C, D, E or F, depending on the type of key desired.

[6] The key exchange method specified for the protocol is one of the key agreement primitives described in SP 800-56A.

K_1 = HASH (K || H || X || *session_id*), where *session_id*[7] is a unique identifier for a SSH connection.
K_2 = HASH (K || H || K_1)
K_3 = HASH (K || H || K_1 || K_2)
……
K_N = HASH (K || H || K_1 || K_2 ||…|| $K_{(N-1)}$)

KEY = the L left most bits of (K_1 || K_2 ||…. K_N)

Figure **4** below shows the accumulative and feedback operation of the KDF.



KEY = L left most bits of (K_1 || K_2 ||…$K_{(N-1)}$|| K_N)

Figure 4: SSH KDF

Each box in Figure 4 represents a hash function computation. The variables in the box are input to the hash function. The arrow (→) indicates an output from the hash function.

[7] *session_id* is actually H from the first key exchange and remains unchanged for the whole connection. However, K and H will be changed when a key re-exchange is performed, see Section 9 of RFC 4253.

The SSH KDFs are **approved** when the following conditions are satisfied:

(1) They are performed in the context of the SSH protocol.

(2) The hash function is one of the hash functions specified in FIPS 180-3.

(3) The hash function deployed in the KDFs meets the security strength(s) required by the cryptographic function(s) for which the keying material is being generated. The security strengths of **approved** hash functions used in KDFs can be found in SP 800-57.

5.3 The Secure Real-time Transport Protocol (SRTP) Key Derivation Function

The SRTP is specified in RFC 3711. The protocol is intended to provide confidentiality, message authentication, and replay protection to the Real-time Transport Protocol (RTP) traffic and to the control traffic for RTP: the Real-time Transport Control Protocol (RTCP) (RFC 3550 [RFC 3550]). Session encryption keys (for a confidentiality service), cipher/encryption salts and authentication keys (for message authentication) in the SRTP are derived from a single master key, called *k_master*, using a KDF. Note that *k_master* is used as a key derivation key when input to the KDF. Details of the KDF can be found in Sections 4.3, 5.3 and 7.1 of RFC 3711.

Denote the cryptographic key (encryption key, cipher salt or authentication key (HMAC key), etc…) to be derived as *K*. The length of *K* is denoted by *L*. Below is a description of the KDF.

master_salt: a random non-secret value.

kdr: the key derivation rate. *kdr* is a number from the set $\{0, 1, 2^1, 2^2, 2^3 \ldots, 2^{24}\}$.

index: a 48-bit value in RTP or a 32-bit value in RTCP. See Sections 3.2.1 and 4.3.2 of RFC 3711 for details.

A function, *DIV*, is defined as followed:

a and *x* are non-negative integers.

$a \; DIV \; x = \lfloor a \mid x \rfloor$. ($a \; DIV \; x$) is represented as a bit string whose length (in bits) is the same as *a*.

label: an 8-bit value represented by two hexadecimal numbers from the set of {0x00, 0x01, 0x02, 0x03, 0x04, 0x05}. In the future, this set might also include any or all of the values 0x06, 0x07,…,0xff.

$$key\_id = label \,||\, (index \; DIV \; kdr)$$

$input = (key\_id \oplus master\_salt) \| 0x0000$,

where *key_id* and *master_salt* are right-aligned. More zero bits are pre-pended to the *key_id* so that len(*master_salt*) = len(*key_id*) before the $(key\_id \oplus master\_salt)$ operation is performed. After *key_id* and *master_salt* are *XOR*ed together, the result is then appended with 16 zero bits to form the *input*, a 128-bit value.

The *input* is used as input to AES-128, AES-192 or AES-256 to produce a session encryption key, cipher salt or authentication key, with *k_master* being the encryption key. *k_master* is 128, 192 or 256 bits, depending on which encryption function is used. More details can be found in RFC 6188 [RFC 6188].

$m = \lceil L/128 \rceil$ (*128* is the bit length of the AES output, and $m \geq 1$)

$Derived\_keying_1 = \text{AES}(k\_master, input)$

$Derived\_keying_2 = \text{AES}(k\_master, input + 1)$

.

.

.

$Derived\_keying_m = \text{AES}(k\_master, input + (m - 1))$

K = the L leftmost bits of

$(Derived\_keying_1 \| Derived\_keying_2 \| \ldots \| Derived\_keying_m)$.

The SRTP KDF is **approved** when the following conditions are satisfied:

(1) The KDF is performed in the context of the SRTP protocol.
(2) The AES encryption operation is as specified in FIPS 197.

5.4 Simple Network Management Protocol (SNMP) Key Derivation Function/Key Localization Function

The User-based Security Model (USM) for SNMP version 3 (SNMPv3) (RFC 2571 [RFC 2571]) is specified in RFC 2574. In this security model, a key localization function (i.e., a key derivation function) is used with a secret password when a user needs to share a different secret key with each authoritative SNMP engine[8]. Two key localization functions are specified in this model. Each uses an MD5 or SHA-1 hash function to generate different secret keys to share with each authoritative SNMP engine. The inputs to the key localization function are the password and the *snmpEngineID*, which is unique for each authoritative SNMP engine. If the hash function is collision resistant, then by using different *snmpEngineID*s, the key localization function will produce different keys. Below is the description of the key localization function with SHA-1.

[8] One of the SNMP engines involved in each communication is designated to be the authoritative SNMP engine, see RFC 2574 Section 1.5.1 for details.

Denote *engineLength* and *passwordlen* to be the lengths (in bytes) of an *snmpEngineID* and a *password*, respectively.

Let $N = \lceil 1048576 / passwordlen \rceil$.

Expanded_password = the leftmost 1048576 bytes of the string of N repetitions of the *password*.

Derived_password = SHA-1 (*Expanded_password*). The *Derived_password* is the output of hashing the *Expanded_password* by SHA-1.

Let *Shared_key* to be the key that the user shares with the authoritative SNMP engine with ID *snmpEngineID*. The *Shared_key* is generated as follow:

Shared_key = SHA-1(*Derived_password* || *snmpEngineID* || *Derived_password*).

The USM KDF is **approved** when the following conditions are satisfied:

(1) It is performed in the context of the SNMP protocol.

(2) SHA-1 (as specified in FIPS 180-3) is used in the key localization function.

5.5 Trusted Platform Module (TPM) Key Derivation Function

Version 1.2 of the Trusted Platform Module is specified in the TPM Main Specification Parts 1, 2 and 3 ([TPM Principles], [TPM Structures] and [TPM Commands], respectively). It uses a SHA-1 HMAC-based Pseudorandom Function (PRF) in its KDF to generate keying material for transport sessions between the TPM and an application running on another processor.

To protect the integrity of communications, a session key, *SKEY*, is derived from a secret authorization value, *Auth*, that is a secret key shared between the TPM and the application. An HMAC-PRF is used to produce the *SKEY* as follows:

> *SKEY* = HMAC (*Auth*, *Nonce_even* || *Nonce_odd*), where *Nonce_even* and *Nonce_odd* are non-secret values created by the random number generators on the TPM and the application, respectively.

Additional technical details for the variables in the functions are provided in Part 3 of the TPM Main Specification.

SKEY is used as an HMAC key to provide data integrity for communications during the session. Further keys may be derived from *SKEY* to provide encryption of parts of the session.

The TPM KDF is **approved** when the following conditions are satisfied:

(1) The TPM KDF is performed in the context of a TPM session (i.e. performed between a TPM and an application with a shared authorization value.)