The NIST SP 800-135 Existing Application-Specific Key Derivation Function Validation System (ASKDFVS)

Updated: October 01, 2012
Original: March 23, 2012

Timothy A. Hall

National Institute of Standards and Technology

Information Technology Laboratory

Computer Security Division

TABLE OF CONTENTS

Update Log

10/01/12

- Section 6.6 SSH KDF: Added definition of ‘K’ as being of type ‘mpint’. Changed example input values.
- Increased the number of trials for all KDFs.

1 Introduction

This document, *The NIST SP 800-135 Existing Application-Specific Key Derivation Function Validation System (ASKDFVS)*, specifies the procedures involved in validating implementations of the key derivation functions approved in NIST SP 800-135, *Recommendation for Existing Application-Specific Key Derivation Functions (December 2011)* [1], namely, the IKEv1, IKEv2, TLS, ANS X9.63-2001, SSH, SRTP, SNMP, and TPM KDFs.

The *ASKDFVS* is designed to perform automated testing on Implementations Under Test (IUTs). This document provides the basic design and configuration of the A*SKDFVS*. It defines the purpose, the design philosophy, and the high-level description of the validation process for the existing application-specific KDFs identified in NIST SP 800-135. The requirements and procedures to be followed by those seeking formal validation of an implementation of one of these KDFs are presented. The requirements described include the specification of the data communicated between the IUT and the ASKDFVS, the details of the tests that the IUT must pass for formal validation, and general instruction for interfacing with the ASKDFVS.

A set of KDF test vectors is available on the http://csrc.nist.gov/groups/STM/cavp/index.html website for testing purposes.

2 Scope

This document specifies the tests required to validate IUTs for conformance to the application-specific KDFs in NIST SP 800-135 [1]. Each KDF specified in NIST SP 800-135 can be validated as an individual component. When applied to IUTs that implement a KDF, the ASKDFVS provides testing to determine the correctness of that algorithm contained in the implementation. The ASKDFVS consists of a single test for each application-specific KDF that determines if the KDF implementation produces the expected keying material outputs given a set of secret and non-secret inputs.

3 Conformance

The successful completion of the individual validation test contained within the ASKDFVS that pertains to the application-specific algorithm being validated is required to claim conformance to that application-specific algorithm in NIST SP 800-135. Testing for the cryptographic module in which the application-specific algorithm is implemented is defined in FIPS PUB 140-2, *Security Requirements for Cryptographic Modules* [2].

4 Definitions and Abbreviations

4.1 Definitions

DEFINITION	**MEANING**
CST laboratory	Cryptographic Security Testing laboratory that operates the ASKDFVS
Key Derivation Function	A function for generating keying material

4.2 Abbreviations

ABBREVIATION	**MEANING**
ANS	American National Standard
ASKDFVS	Application-Specific Key Derivation Function Validation System
FIPS	Federal Information Processing Standard
HMAC	Keyed-Hash Message Authentication Code
ISAKMP	Internet Security Association and Key Management Protocol
IUT	Implementation Under Test
IKE	Internet Key Exchange
KDF	Key Derivation Function
SA	ISAKMP Security Association
SNMP	Simple Network Management Protocol
SRTP	Secure Real-time Protocol
SSH	Secure Shell
TLS	Transport Layer Security
TPM	Trusted Platform Module

5 Design Philosophy of the Existing Application-Specific Key Derivation Functions System

The ASKDFVS is designed to test conformance to each individual application-specific KDF specified in NIST SP 800-135 rather than provide a measure of a product's security. The validation tests are designed to assist in the detection of accidental implementation errors, and are not designed to detect intentional attempts to misrepresent conformance. Thus, validation should not be interpreted as an evaluation or endorsement of overall product security.

The ASKDFVS has the following design philosophy:

1. The ASKDFVS is designed to allow the testing of an IUT at locations remote to the ASKDFVS. The ASKDFVS and the IUT communicate data via *REQUEST (.req)* and *RESPONSE (.rsp)* files. The ASKDFVS also generates *SAMPLE (.sam)* files to provide the IUT with an example of the *RESPONSE* file format.

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

6 ASKDFVS Tests

When applied to an IUT, the ASKDFVS provides testing to determine the correctness of each individual application-specific KDF. Each application-specific KDF specified in NIST SP 800-135 is tested individually. The ASKDFVS consists of a single validation test for each application-specific KDF. The ASKDFVS requires the vendor to specify the application-specific KDF to be tested (e.g., IKEv1, IKEv2, TLS 1.0/1.1, etc.). Separate files will be generated for each KDF. In the case of the IKE v1 KDF, a separate file is generated for each authentication method supported.

6.1 Configuration Information

To initiate the validation process of the ASKDFVS, a vendor submits an application to an accredited laboratory requesting the validation of its implementation of one or more application-specific KDF. The vendor's implementation is referred to as the Implementation Under Test (IUT). The request for validation includes background information describing the IUT along with information needed by the ASKDFVS to perform the specific tests. More specifically, the request for validation includes:

1. Cryptographic algorithm implementation information

    a) Vendor Name

    b) Implementation Name;

    c) Implementation Version;

d) Indication if implemented in software, firmware, or hardware;

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

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

2. Configuration information for the ASKDFVS tests

   a) The application specific KDFs implemented:

      i. IKEv1 KDF

      ii. IKEv2 KDF

      iii. TLS KDF (includes TLS 1.0/1.1 KDF and TLS 1.2 KDF)

      iv. ANS X9.63-2001 KDF

      v. SSH KDF

      vi. SRTP KDF

      vii. SNMP KDF

      viii. TPM KDF

3. If IKEv1 KDF implemented:

   a) Authentication methods used:

      i. Digital signature authentication

      ii. Public key encryption authentication

      iii. Pre-shared key authentication

   b) Well-known group(s) supported (for determining shared secret length)

      i. Group 2: 1024 bit MODP

      ii. Group 4: 185 bit EC2N

      iii. Group 14: 2048 bit MODP

   c) SHA functions supported for each group

      i. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 for Group 2

    ii. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 for Group 4

    iii. SHA-224, SHA-256, SHA-384, SHA-512 for Group 14

  d) Two supported lengths for initiator and responder's nonce

    i. Must range between 64 bits (8 bytes) and 2048 bits (256 bytes)

  e) If pre-shared key authentication is supported

    i. Minimum and maximum pre-shared key length

4. If IKEv2 KDF implemented:

  a) Well-known groups supported (for determining shared secret length)

    i. Group 2: 1024 bit MODP

    ii. Group 4: 185 bit EC2N

    iii. Group 14: 2048 bit MODP

  b) SHA functions supported for each group

    i. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 for Group 2

    ii. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 for Group 4

    iii. SHA-224, SHA-256, SHA-384, SHA-512 for Group 14

  c) Two supported lengths for initiator and responder's nonce

    i. Must range between 64 bits (8 bytes) and 2048 bits (256 bytes)

  d) Two bit lengths of derived keying material (output of the key expansion step)

  e) Two bit lengths of derived keying material for Child SA

5. If TLS KDF implemented, which TLS version(s) implemented:

  a) TLS 1.0/1.1, TLS 1.2

  b) If TLS 1.2 implemented, indicate SHA functions supported

    i. SHA-256, SHA-384, SHA-512

6. If ANS X9.63-2001 KDF implemented:

  a) Minimum and maximum elliptic curve field size supported:

i. Must be a NIST Elliptic Curve

ii. This determines the length of the shared secret generated by CAVS

b) SHA functions supported

i. Only SHA functions allowed for use with EC field size according to NIST SP 800-56A Table 2.

c) Two bit lengths of output key material (key data).

i. One should be shortest valid derived key length

ii. Other longest supported key data length, up to 4096 bits

7. If SSH KDF implemented:

   a) Diffie-Hellman shared secret lengths supported. Only 1024 and 2048 bits are valid values.

   b) SHA functions supported:

      i. SHA-1, SHA-224, SHA-256, SHA-384, SHA-512

   c) Cipher supported. Used to determine length of output IV and key values.

      i. All implementations must support TDES CBC. CAVS tests include 64 bit IV (TDES block length) and 192 bit key (3-key TDES key length) outputs

      ii. Optional AES CBC. If AES not supported, do not check box. If more than one AES function (key length) supported, choose any one.

8. If SRTP KDF implemented:

   a) AES cipher functions (key lengths) supported:

      i. AES-128, AES-192, or AES-256

9. If SNMP KDF implemented:

   a) Two valid SNMP engine IDs. If implementation only supports one, then use same value twice.

   b) Two valid password lengths, in characters.

10. If TPM KDF implemented, no additional configuration information needed.

6.2 The IKEv1 KDF Test

The IKEv1 KDF test is organized as follows. CAVS generates a separate request (.req) file for each authentication method supported: *ikev1_dsa.req* for digital signature authentication, *ikev1_pke.req* for public key encryption authentication, and *ikev1_psk.req* for pre-shared key authentication.

Each file begins with a header that has the CAVS version on line one, the IKEv1 authentication method and the implementation name on line two, the options tested on line three, and the date and time the file was generated on line four. An option is defined by the Diffie-Hellman shared secret well-known group and SHA function used.

The allowable Diffie-Hellman shared secret groups are the well-known groups in IETF RFC 2409, Section 6 that meet the requirements for FFC, ECC, and IFC Parameters for key establishment found in NIST SP 800-56A, Tables 1 and 2 and NIST SP 800-56B Table 1, respectively. Specifically, these are Group 2, a 1024 bit MODP group, Group 4, a 185 EC2N group, and Group 14, a 2048 bit MODP group.

The allowable SHA functions are any Approved SHA function in FIPS 180-3 that can be used with a given shared secret size according to NIST SP 800-56A, Tables 1 and 2 or NIST SP 800-56B Table 1. Specifically, these are SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 with Group 2 or Group 4, and SHA-224, SHA-256, SHA-384, and SHA-512 with Group 14. As stated above, the options ("Cases tested") are listed on the third line of the request file, for example:

```
# CAVS 12.0
# 'IKE v1 Digital Signature Authentication' information for abc
# Cases tested:  [Group 2, SHA-1] [Group 2, SHA-224] [Group 14, SHA-512]
# Generated on Fri Feb 10 11:24:30 2012
```

For each Group/SHA combination, CAVS generates two sets of 100 trials. In addition to the Diffie-Hellman shared secret (g^xy) length and the SHA function used, the length in bits of the initiator's nonce (Ni) and the responder's nonce (Nr) are given in square brackets:

```
[g^xy length = 1024]
[SHA-1]
[Ni length = 64]
[Nr length = 64]
```

Ni and Nr must be between 64 bits (8 bytes) and 2048 (256 bytes) bits long. Note that the Diffie-Hellman shared secret, g^xy, for Group 4 has a field size of 185 bits but is represented by a full 192 bits (24 bytes), with the 7 most significant bits always 0. So, even though Group 4 cases will have '[g^xy length = 185]', the g^xy values will be expressed as 48 hexadecimal numbers. The full 192 bits are used in the KDF. See the IKEv2 section for an example of Group 4 Diffie-Hellman shared secret values. Each trial, or count, has the following format (Group 2, SHA-1 example):

```
COUNT = 0
CKY_I = 96e0db28659a09a0
CKY_R = 15b1dbf1265e4bb0
Ni = f3f0d1cbb395cca1
Nr = 6a9d709fde7284ea
g^xy =
8b7f07239757d1f80fa5c8df7ec16633ad2c28ee5f4ebc0b2626c9b53d9475186930146fa664ef
1f9457c3ab14c535036fb21c465d4b380c6209e25b12e35570e9380e446d25f033e71bb453098c
d8c77c0eec66bc9c49be2344d799edfc7281a53c54a78c076562e0dfd7fa1bb242084060a3be7f
7d95d35589bdbc8ef93264
```

CKY_I and CKY_R are the initiator’s cookie and the responder’s cookie, respectively. They are always 64 bits (8 bytes) long. CKY_I, CKY_R, Ni, Nr, and g^xy values are represented in hexadecimal. The sample file indicates the outputs that the IUT needs to provide.

```
COUNT = 0
CKY_I = c71b24e6db4b65bc
CKY_R = bd666b0f3f25b7d7
Ni = 813f834a83c7e1d9
Nr = fa26222026cd8739
g^xy =
1ef3c0f105ec528a626a5bcb1f692f25e2e670edd7509b86d21aaa3aacc0fad0440fbe45b7806e
95328fbb720acbf2febe7ca5beec92706eb9edacb666e213f8d97b6a19eb044fa53290eaeea976
cb43c74921e30574e6497ae7b820af01e000654fe6ac111814cc1fa957fd5f414e8dd01beb1621
c84fcfd0992efdcdfb6747
SKEYID = ?
SKEYID_d = ?
SKEYID_a = ?
SKEYID_e = ?
```

The SKEYID value is the result of the extraction step. It is computed using different input values for the different authentication methods [1][3]. Its length is always the output block length of the keyed hash function used (i.e., SKEYID is 160 bits long for HMAC-SHA-1, 256 bits long for HMAC-SHA-256, etc.). SKEYID_d, SKEYID_a, and SKEYID_e are results of the expansion step. They are computed using the same inputs for all three authentication methods and are the same length as SKEYID.

For the case of pre-shared key authentication, there is an additional input, the pre-shared key. The pre-shared key length is in brackets, e.g., ‘[pre-shared key length = 128]’, and a random pre-shared key value is generated for each trial (COUNT).

The ASKDFVS verifies the correctness of the IUT’s values of SKEYID, SKEYID_d, SKEYID_a, and SKEYID_e by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT’s implementation of the IKEv1 KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

6.3 The IKEv2 KDF Test

The IKEv2 KDF test is organized as follows. CAVS generates a single request file, *ikev2.req*, for the IKEv2 tests.

The file begins with a header that has the CAVS version on line one, ‘IKEv2’ and implementation name on line two, the options tested on line three, and the date and time the file was generated on line four. An option is defined by the Diffie-Hellman shared secret well-known group and SHA function used. These group/SHA function options are defined and composed in the same way that the IKEv1 options are. See the above section for details on these options.

A sample IKEv2 file header is shown below.

```
# CAVS 12.0
# 'IKE v2' information for abc
# Cases tested: [Group 4, SHA-1] [Group 4, SHA-224] [Group 14, SHA-224]
# Generated on Tue Feb 07 15:40:24 2012
```

For each Group/SHA combination, CAVS generates two sets of 100 trials. In addition to the Diffie-Hellman shared secret (g^ir) length and the SHA function used, the length in bits of the initiator’s nonce (Ni) and the responder’s nonce (Nr) are given in square brackets. Also, the output lengths for the expansion steps are given as derived keying material (DKM) length for the parent security authority (SA) and any child SA.

```
[g^ir length = 185]
[SHA-1]
[Ni length = 64]
[Nr length = 64]
[DKM length = 1056]
[Child SA DKM length = 1056]
```

Ni and Nr must be between 64 bits (8 bytes) and 2048 (256 bytes) bits long. Note that the Diffie-Hellman shared secret, g^ir, for Group 4 has a field size of 185 bits but is represented by a full 192 bits (24 bytes), with the 7 most significant bits always 0. So, even though Group 4 cases will have ’[g^ir length = 185]’, the g^ir values will be expressed as 48 hexadecimal numbers. The full 192 bits are used in the KDF. Each trial, or count, has the following format (Group 4, SHA-1 example):

```
COUNT = 0
Ni = afcd6b485c41095c
Nr = c6b9221c87ae8894
g^ir = 0130bf62d1b7ab9ad0f0edc293f15fb1442ee98197c450d0
g^ir (new) = 00504d95b4220f9577603a3cd58d240babd34fdebbd7d97a
SPIi = 1cde96e074d1179a
SPIr = 7b69ba7f9564d2cb
```

SPIi and SPIr are the security parameter indices of the initiator and the responder, respectively. They are always 64 bits (8 bytes) long. SPIi, SPIr, Ni, Nr, and g^ir values are represented in hexadecimal. The sample file indicates the outputs that the IUT needs to provide.

```
COUNT = 0
Ni = afcd6b485c41095c
Nr = c6b9221c87ae8894
g^ir = 0130bf62d1b7ab9ad0f0edc293f15fb1442ee98197c450d0
g^ir (new) = 00504d95b4220f9577603a3cd58d240babd34fdebbd7d97a
SPIi = 1cde96e074d1179a
SPIr = 7b69ba7f9564d2cb
SKEYSEED = ?
DKM = ?
DKM(Child SA) = ?
DKM(Child SA D-H) = ?
SKEYSEED(Rekey) = ?
```

The SKEYSEED value is the result of the extraction step. Its length is always the output block length of the keyed hash function used (i.e., SKEYSEED is 160 bits long for HMAC-SHA-1, 256 bits long for HMAC-SHA-256, etc.). DKM stands for "derived keying material" and is the result of the expansion step. DKM(Child SA) and DKM(Child SA D-H) are the expansion step results for a child security authority (SA) as computed in Section 2.17 of the IKEv2 RFC [4], with and without a new Diffie-Hellman shared secret (i.e., g^ir (new)), respectively. SKEYSEED(Rekey) is the new SKEYSEED computed as in Section 2.18 of the IKEv2 RFC using g^ir (new).

The ASKDFVS verifies the correctness of the IUT's values of SKEYSEED, DKM, DKM (Child SA), DKM (Child SA), and SKEYID (Rekey) by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT's implementation of the IKEv2 KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

6.4 The TLS KDF Test

The TLS KDF test is organized as follows. CAVS generates a single request file, *tls.req*, for the TLS tests.

The file begins with a header that has the CAVS version on line one, 'TLS' and implementation name on line two, the options tested on line three, and the date and time the file was generated on line four. An option is defined by the version of TLS used, either TLS 1.0/1.1 or TLS 1.2 and SHA function used. TLS 1.0/1.1 only uses SHA-1. TLS 1.2 can use SHA-256, SHA-384, and SHA-512.

A sample TLS file header is shown below.

```
# CAVS 12.0
# 'TLS' information for abc
# Cases tested:  [TLS 1.0/1.1] [TLS 1.2, SHA-256] [TLS 1.2, SHA-512]
# Generated on Thu Feb 09 16:24:01 2012
```

For each option, CAVS generates one set of 100 trials. In addition to the TLS version and, for TLS 1.2, the SHA function used, the pre-master secret length in bits is given. Also, the output length for the expansion step is given as the key block length, in bits.

```
[TLS 1.2, SHA-256]
[pre-master secret length = 384]
[key block length = 1024]
```

Each trial in the request file has the following format:

```
COUNT = 0
pre_master_secret =
083935f066ea992f0e631c3711a2ca035dba1667882b062f0e0d065e6b2aee56d35ece5111c97f
49de67bcdf136a4e6d
serverHello_random =
d013c5535eb1fa899caa429e5329ba54dd801ed01ed4ab363cab65a59a3b5916
clientHello_random =
99445a3f1eb792535f741620c51ec8bb1be9e4f49e8e7fc619b90e9dc1b65a94
server_random =
5946b7c8cb3aa2b37763059e4efdfb0c42e2ffe831f59a57b9e47b5d82d9333d
client_random =
8c19baa70d49fc17dcd839cffed3d5c31e25784ec0add80495b79da674d83869
```

In addition to the pre-master secret, two sets of server and client random values are given. The first set is from the “Hello” message and is used in the computation of the master secret in the extraction step. The second pair is used in the key expansion step to compute a key block. The sample file format shows the two outputs: master secret, a 384 bit (48 byte) value, and the key block output of the expansion step, length specified in case header (e.g., “[key block length = 1024]”).

```
COUNT = 0
pre_master_secret =
083935f066ea992f0e631c3711a2ca035dba1667882b062f0e0d065e6b2aee56d35ece5111c97f
49de67bcdf136a4e6d
serverHello_random =
d013c5535eb1fa899caa429e5329ba54dd801ed01ed4ab363cab65a59a3b5916
clientHello_random =
99445a3f1eb792535f741620c51ec8bb1be9e4f49e8e7fc619b90e9dc1b65a94
server_random =
5946b7c8cb3aa2b37763059e4efdfb0c42e2ffe831f59a57b9e47b5d82d9333d
client_random =
8c19baa70d49fc17dcd839cffed3d5c31e25784ec0add80495b79da674d83869
master_secret = ?
key_block = ?
```

The ASKDFVS verifies the correctness of the IUT’s values of master_secret and key_block by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT's implementation of the TLS KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

6.5 The ANS X9.63-2001 KDF Test

The ANS X9.63-2001 KDF test is organized as follows. CAVS generates a single request file, *ansx963_2001.req*, for the ANS X9.63-2001 tests.

The file begins with a header that has the CAVS version on line one, 'ANS X9.63-2001' and implementation name on line two, the options tested on line three, and the date and time the file was generated on line four. An option is defined by the SHA function used. A sample ANS X9.63-2001 file header is shown below.

```
# CAVS 12.0
# 'ANS X9.63-2001' information for abc
# SHA sizes tested: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
# Generated on Fri Mar 09 14:52:11 2012
```

For each option, CAVS generates two sets of 100 trials. In addition the SHA function used, each set (or case) identifies the length of the shared secret (Z) in bits, the length of the SharedInfo, which can be zero, in bits, and the length of the key data generated (i.e., the output of the KDF) in bits. The valid lengths of the shared secret, Z, are the field sizes of the NIST curves, namely 163, 233, 283, 409, and 571 bits for binary (B) and Koblitz (K) curves and 192, 224, 256, 384, and 521 for the prime (P) curves. The user is asked to provide the shortest and longest field size supported by the implementation. No curve computations are performed; the field size is only used for the length of Z. If a field size that is not a multiple of 8 bits is used, such as 233 bits or 521 bits, the shared secret is represented by a string of bytes (octets) that is padded with zeros on the left (i.e., most significant bits) to fill out a full byte (octet). Thus, the shared secret (Z) value for a field size 233 would be represented by 240 bits (30 bytes), with the 7 leftmost bits as zeros. Below is an example for shared secret (Z) length of 192 bits, corresponding to use of the P-192 NIST curve:

```
[SHA-1]
[shared secret length = 192]
[SharedInfo length = 0]
[key data length = 128]
```

Each trial in the request file has the following format:

```
COUNT = 0
Z = 64ec68a1a0d55bb94bd43d0321f3d52314d53010c4878b8c
SharedInfo =
```

Note the format of the zero-length SharedInfo input. There is a single output value, key_data, as shown in sample file format:

```
COUNT = 0
Z = 64ec68a1a0d55bb94bd43d0321f3d52314d53010c4878b8c
SharedInfo =
key_data = ?
```

The ASKDFVS verifies the correctness of the IUT's values of key_data by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT's implementation of the ANS X9.63-2001 KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

6.6 The SSH KDF Test

The SSH KDF test is organized as follows. CAVS generates a single request file, *ssh.req*, for the SSH tests.

The file begins with a header that has the CAVS version on line one, 'SSH' and implementation name on line two, the options tested on line three, and the date and time the file was generated on line four. An option is defined by the SHA function used. A sample SSH file header is shown below.

```
# CAVS 14.1
# 'SSH' information for ssh_example
# SHA sizes tested: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
# Generated on Mon Sep 24 14:26:51 2012
```

For each option, CAVS generates two sets of 100 trials. In addition to the SHA function used, each set (or case) identifies the length of the shared secret (K) in bits, the length in bits of the output IV values, and the length in bits of the output encryption keys. The valid lengths of the shared secret, K, are 1024 and 2048 bits. K is of type 'mpint' and thus has a four byte field at the beginning (left) that indicates the length of the key in bytes. Also, if the most significant bit of the raw K value (not the length field) is a '1', a single zero byte is pre-pended to the value of K. The lengths of the IVs and encryption keys are determined by the block ciphers supported. IV lengths are determined by the cipher block size, and encryption key lengths are determined by the cipher key lengths. All implementations must support 3-key TDES CBC, which has a block size of 64 bits and a key length of 192 bits. If the implementation supports one or more AES ciphers (key lengths), then one of them will be used for a case with a 128 bit IV length and a 128, 192, or 256 bit encryption key length.

```
[SHA-1]
[shared secret length = 1024]
[IV length = 64]
[encryption key length = 192]
```

Each trial in the request file has the following format:

```
COUNT = 0
K=00000080631236da998554b09188930ccfc5445654afe4242b02e671cf2db5812677361f5dd1
a148bee9db299d930003c6027bbf9502354f91f9a7504e0d6dfedfc1a07b8699d068cef0fec392
19914c41c9cf19ecbc81f53b07bb5129a6f7f8807b943922550be9887981ae5ce85da7aa28e685
6839d0322bafa175153c72dbe02ff831
H = 744b0bb7ad150f54819128a56953614b62e88c0c
session_id = 744b0bb7ad150f54819128a56953614b62e88c0c
```

Note that H and session_id are the same length as the SHA function output block. Also, since session_id is the exchange hash from the first key exchange, H and session_id will be equal in some trials. There are six output values calculated for each trial, one for each value of “A” through “F” as defined in IETF RFC 4253 Section 7.2, “Output from Key Exchange” [9]. An example is shown below:

```
COUNT = 0
K =
00000080631236da998554b09188930ccfc5445654afe4242b02e671cf2db5812677361f5dd1a1
48bee9db299d930003c6027bbf9502354f91f9a7504e0d6dfedfc1a07b8699d068cef0fec39219
914c41c9cf19ecbc81f53b07bb5129a6f7f8807b943922550be9887981ae5ce85da7aa28e68568
39d0322bafa175153c72dbe02ff831
H = 744b0bb7ad150f54819128a56953614b62e88c0c
session_id = 744b0bb7ad150f54819128a56953614b62e88c0c
Initial IV (client to server) = ?
Initial IV (server to client) = ?
Encryption key (client to server) = ?
Encryption key (server to client) = ?
Integrity key (client to server) = ?
Integrity key (server to client) = ?
```

The ASKDFVS verifies the correctness of the IUT’s values of key_data by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT’s implementation of the SSH KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

6.7 The SRTP KDF Test

The SRTP KDF test is organized as follows. CAVS generates a single request file, *srtp.req*, for the SRTP tests.

The file begins with a header that has the CAVS version on line one, ‘SRTP’ and implementation name on line two, the options tested on line three, and the date and time the file was generated on line four. An option is defined by the AES function (i.e., key length) used. A sample SRTP file header is shown below.

```
# CAVS 12.0
# 'SRTP' information for abc
# AES key lengths tested: AES-128, AES-192, AES-256
# Generated on Wed Mar 07 16:32:42 2012
```

For each option, CAVS generates a set of 100 trials. The trials begin after the option (or case) header:

```
[AES-128]
```

Each trial in the request file has the following format:

```
COUNT = 0
k_master = dcd03a3d4f6f499d546039b21b2fdd29
master_salt = 1e7a09ba652761146f175d453317
kdr = 000000000000
index = 8b92fb0f0de4
index (SRTCP) = 25fe977f
```

The KDF has five inputs. A single master key, k_master, has length equal to the AES key length for the option (i.e., k_master is 128 bits long for AES-128 option, 192 bits long for AES-192 option, and 256 bits long for the AES-256 option). The master_salt is a random non-secret value, always 112 bits long. The key derivation rate, kdr, is a number in the set {0, 1, 2, 4, 8, 16, …, 2^{24}}. In the request and sample files, it will have extra, non-significant zeros that do not impact calculations, since it is a number and not a string of bits. Two values are given for the index: ‘index’ is a 48 bit index value used in the SRTP kdf; ‘index (SRTCP)’ is a 32-bit index used in the SRTCP kdf. When computing the output of the SRTCP kdf, use the same k_master, master_salt, and kdr as in the SRTP kdf for this trial. The fifth input is label, an 8 bit value in the set {0x00, 0x01, 0x02, 0x03, 0x04, 0x05}. All six values of label are used in each trial. The sample file indicates the six output keys calculated per trial.

```
COUNT = 0
k_master = dcd03a3d4f6f499d546039b21b2fdd29
master_salt = 1e7a09ba652761146f175d453317
kdr = 000000000000
index = 8b92fb0f0de4
index (SRTCP) = 25fe977f
SRTP k_e = ?
SRTP k_a = ?
SRTP k_s = ?
SRTCP k_e = ?
SRTCP k_a = ?
SRTCP k_s = ?
```

The six outputs calculated per trial (defined in [10]) are as follows. The SRTP encryption key (STRP k_e) is calculated using label = 0x00 and the 48 bit index and is the same length as the AES key length for the option, i.e., 128 bits for AES-128. The SRTP authentication key (SRTP k_a) is calculated using label = 0x01 and the 48 bit index and is always 160 bits. The SRTP salting key (SRTP k_s) is calculated using label = 0x02 and the 48 bit index and is always 112 bits. SRTCP encryption key (SRTCP k_e) uses label = 0x03, the 32-bit SRTCP index, and is the same length as SRTP k_e. The SRTCP authentication key (SRTCP k_a) uses label = 0x04, the 32-bit SRTCP index, and is the same length as SRTP k_a, 160 bits. The SRTCP salting key

(SRTCP k_s) uses label = 0x05, the 32-bit SRTCP index, and is the same length as SRTP k_s, 112 bits.

The ASKDFVS verifies the correctness of the IUT's values of SRTP k_e, SRTP k_a, SRTP k_s, SRTCP k_e, SRTCP k_a, and SRTCP k_s by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT's implementation of the SRTP and SRTCP KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

6.8 The SNMP KDF Test

The SNMP KDF test is organized as follows. CAVS generates a single request file, *snmp.req*, for the SNMP tests.

The file begins with a header that has the CAVS version on line one, 'SNMP' and implementation name on line two, 'SHA-1' on line three, and the date and time the file was generated on line four. A sample SNMP file header is shown below.

```
# CAVS 12.0
# 'SNMP' information for abc
# SHA-1
# Generated on Wed Mar 07 16:25:03 2012
```

CAVS generates two sets of 100 trials for SNMP. Each set of 100 trials has a header with two parameters: engine ID and password length:

```
[engineID = 000002b87766554433221100]
[passwordLen = 8]
```

The engine ID is supplied by the lab tester or vendor. If the implementation under test (IUT) supports more than one engine ID, choose any two valid engine IDs. If the IUT only supports a single engine ID, perhaps hard-wired or hard-coded in the implementation, then use the same value twice. The tester or vendor should choose two different valid password lengths if the IUT supports variable password lengths. If only a single length password is supported, then enter the same length twice. Each trial consists of a single input, a randomly generated character-only password. The same engine ID is used for each trial.

```
COUNT = 0
password = PVpTTkYg
```

The output is the 160 bit Shared key, as indicated in the sample file:

```
COUNT = 0
password = PVpTTkYg
```

```
Shared_key = ?
```

The ASKDFVS verifies the correctness of the IUT's values of Shared_key by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT's implementation of the SNMP KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

6.9 The TPM KDF Test

The TPM KDF test is organized as follows. CAVS generates a single request file, *tpm.req*, for the TPM tests.

The file begins with a header that has the CAVS version on line one, 'TPM' and implementation name on line two, 'HMAC-SHA-1' on line three and the date and time the file was generated on line four. A sample TPM file header is shown below.

```
# CAVS 12.0
# 'TPM' information for abc
# HMAC-SHA-1
# Generated on Wed Mar 07 16:25:04 2012
```

CAVS generates one set of 100 trials for TPM. There is no header. All three inputs are 160 bits (20 bytes) in length. A trial looks like this

```
COUNT = 0
Auth = 431ff1a1eac4b416c4d70ed786df04ddd6294a85
Nonce_even = 8dcda0d9dd059c663310d104009af8d6d30d5c49
Nonce_odd = 02e144dcf416f580bcdc4c7db2175291a0519d4e
```

Auth is a secret key shared between the TPM and the application, and Nonce_even and Nonce_odd are non-secret values generated randomly by the TPM and application, respectively. For this test, CAVS generates all three values randomly. The sample file indicates the one output value, SKEY:

```
COUNT = 0
Auth = 431ff1a1eac4b416c4d70ed786df04ddd6294a85
Nonce_even = 8dcda0d9dd059c663310d104009af8d6d30d5c49
Nonce_odd = 02e144dcf416f580bcdc4c7db2175291a0519d4e
SKEY = ?
```

The ASKDFVS verifies the correctness of the IUT's values of SKEY by comparing them against values calculated by the CAVS for the same inputs. If the values are the same, then the IUT's implementation of the TPM KDF passes the validation test. If the values do not match, the IUT has an error in it. If an error occurs during validation, the values computed by the CAVS and the IUT are written to the log file. The CST Laboratory can use the information in the log file to assist the vendor in debugging the IUT.

Appendix A References

[1] *Recommendation for Existing Application-Specific Key Derivation Functions*, NIST SP 800-135, National Institute of Standards and Technology, December 2011.

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

[3] D. Harkins, D. Carrel, *Internet Key Exchange (IKE)*, RFC 2409, Internet Engineering Task Force (IETF), November 1998.

[4] C. Kaufman, *Internet Key Exchange (IKEv2) Protocol*, RFC 4306, Internet Engineering Task Force (IETF), December 2005.

[5] T. Dierks, C. Allen, *The TLS Protocol, Version 1.0*, RFC 2246, Internet Engineering Task Force (IETF), January 1999.

[6] T. Dierks, E. Rescorla, *The Transport Layer Security (TLS) Protocol, Version 1.1*, RFC 4346, Internet Engineering Task Force (IETF), April 2006.

[7] T. Dierks, E. Rescorla, The *Transport Layer Security (TLS) Protocol, Version 1.2*, RFC 5246, Internet Engineering Task Force (IETF), August 2008.

[8] ANS X9.63-2001, *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, November 20, 2001.

[9] T. Ylonen, C. Lonvick, *The Secure Shell (SSH) Transport Layer Protocol*, RFC 4253, Internet Engineering Task Force (IETF), January 2006.

[10] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman, *The Secure Real-time Transport Protocol (SRTP)*, RFC 3711, Internet Engineering Task Force (IETF), March 2004.

[11] D. McGrew, *The Use of AES-192 and AES-256 in Secure RTP*, RFC 6188, Internet Engineering Task Force (IETF), March 2011.

[12] U. Blumenthal, B. Wijnen, *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*, RFC 2574, Internet Engineering Task Force (IETF), April 1999.

[13] TPM Main Specification Parts 1, 2, and 3, Version 1.2.