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

Updated: September 16, 2015
Previously Updated: March 3, 2015
Original: March 23, 2012

Sharon Keller

Timothy A. Hall

National Institute of Standards and Technology

Information Technology Laboratory

Computer Security Division

TABLE OF CONTENTS

Update Log

9/16/15

- IKEv2 KDF: Minimum length of IKEv2 nonces and Payload is 128 bits NOT 64. As stated in RFC 4306, part 2.10 (Nonces): “Nonces used in IKEv2 MUST be randomly chosen, MUST be at least 128 bits in size …”. RFC 4306, in Section 3.9 (Payload), also notes that, “The size of a Nonce MUST be between 16 and 256 octets inclusive.” which would be between 128 and 2,048 bits."

- Updated text to reflect changes made as a result of SP800-131A. Affected IKEv1, IKEv2, ANS X9.63-2001, and SSH. These changes have been available with the CAVS tool since SP800-131A went into effect January 1, 2014.

3/3/15

- SRTP: Added ability to test specific KDRs. Originally the validation test automatically tested all possible values for KDR. This didn’t work for implementations that only support one KDR value. Changes made to Section 6.1 and 6.7.

9/05/13

- IKEv1 and IKEv2 KDF: More options for Diffie-Hellman shared secret lengths. No longer confined to Groups 2, 4, and 14. Also, now identified by length in bits, not by group.

- SSH KDF: TDES-CBC cipher no longer required. Tester now selects all supported ciphers out of TDES-CBC, AES-128-CBC, AES-192-CBC, AES-256-CBC.

9/25/12

- Section 6.6 SSH KDF: Added definition of ‘K’ as being of type ‘mpint’. Changed example input values.

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) One to three supported Diffie-Hellman shared secret lengths

      i. The shortest shared secret length supported.

      ii. If more than one shared secret length supported, the longest one.

      iii. If more than two shared secret lengths supported, any length not used in (i) or (ii).

   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-1, 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) One to three supported Diffie-Hellman shared secret lengths
        i. The shortest shared secret length supported.
        ii. If more than one shared secret length supported, the longest one.
        iii. If more than two shared secret lengths supported, any length not used in (i) or (ii).
    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-1, 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 128 bits (16 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) Only 2048 bit Diffie-Hellman shared secret length is supported.

   b) SHA functions supported:

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

   c) Block ciphers and key lengths supported. Used to determine length of output IV and key values.

      i. TDES CBC, AES-128 CBC, AES-192 CBC, AES-256 CBC

8. If SRTP KDF implemented:

   a) AES cipher functions (key lengths) supported:

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

   b) Indicate the KDR rates that are supported by the IUT:

      i. All KDRs (0, 1, 2, 4, 8, 16, … 2^24)

      ii. Selected set of KDRs

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 length and SHA function used.

The allowable Diffie-Hellman shared secret lengths are those of the well-known groups in IETF RFCs 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. The allowable SHA functions are any Approved SHA function in FIPS 180-3. Specifically, these are SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. As stated above, the options (“Cases tested”) are listed on the third line of the request file, for example:

```
# CAVS 17.8
# 'IKE v1 Digital Signature Authentication' information for ABC
# Cases tested:  [g^xy length=224, SHA-224] [g^xy length=2048, SHA-1]
# Generated on Wed Sep 16 09:29:33 2015
```

For each Group/SHA combination, CAVS generates two sets of 10 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 = 224]
[SHA-224]
[Ni length = 64]
[Nr length = 64]
```

Ni and Nr must be between 64 bits (8 bytes) and 2048 (256 bytes) bits long. If a field size that is not a multiple of 8 bits is used, such as 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). For example 521 bits is represented by a full 528 bits (66 bytes), with the 7 most significant bits always 0. See the IKEv2 section for an example of Group 4 Diffie-Hellman shared secret values. Each trial, or count, has the following format (g^xy length = 224, SHA-224 example):

```
COUNT = 0
CKY_I = d259fb330129957c
CKY_R = 971f3ac482febda1
```

```
Ni = b24b3aa3a0c11db5
Nr = 27f29e60773ac63c
g^xy = cae246ad4ea0d9b3c1f657b4aeea88eb8c69155f2dcf19ccc97235d1
```

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 = d259fb330129957c
CKY_R = 971f3ac482febda1
Ni = b24b3aa3a0c11db5
Nr = 27f29e60773ac63c
g^xy = cae246ad4ea0d9b3c1f657b4aeea88eb8c69155f2dcf19ccc97235d1
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 length and SHA function used. These length/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 17.8
# 'IKE v2' information for ABC
# Cases tested:  [g^ir length=256, SHA-256]
# Generated on Wed Sep 16 08:35:26 2015
```

For each length/SHA combination, CAVS generates two sets of 10 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 = 256]
[SHA-256]
[Ni length = 128]
[Nr length = 128]
[DKM length = 1056]
[Child SA DKM length = 1056]
```

Ni and Nr must be between 128 bits (16 bytes) and 2048 (256 bytes) bits long. If a field size that is not a multiple of 8 bits is used, such as 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). For example 521 bits is represented by a full 528 bits (66 bytes), with the 7 most significant bits always 0. Each trial, or count, has the following format (Group 4, SHA-256 example):

```
COUNT = 0
Ni = c17a5f006ecc2ba06bedd0aa16c96449
Nr = fd0eb2688b8d4ced8c774ff7f86397d8
g^ir = 62986199e52d6d7adebe21f902ee6f9f7d856821a54050b0baf67a9c789f3919
g^ir (new) = f70c74c3c46f30063b923cf32a0e8597f0f29766fbdfbb1a8835b985c8ae6a7e
SPIi = a3ae853f76747f0b
SPIr = 59a91c9738b6a80b
```

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 = c17a5f006ecc2ba06bedd0aa16c96449
Nr = fd0eb2688b8d4ced8c774ff7f86397d8
g^ir = 62986199e52d6d7adebe21f902ee6f9f7d856821a54050b0baf67a9c789f3919
g^ir (new) = f70c74c3c46f30063b923cf32a0e8597f0f29766fbdfbb1a8835b985c8ae6a7e
SPIi = a3ae853f76747f0b
SPIr = 59a91c9738b6a80b
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 10 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 17.8
# 'ANS X9.63-2001' information for ABC
# SHA sizes tested: SHA-224, SHA-256, SHA-384, SHA-512
# Generated on Wed Sep 16 09:42:52 2015
```

For each option, CAVS generates two set of 10 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 233, 283, 409, and 571 bits for binary (B) and Koblitz (K) curves and 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 224 bits, corresponding to use of the P-224 NIST curve:

```
[SHA-224]
[shared secret length = 224]
[SharedInfo length = 0]
[key data length = 128]
```

Each trial in the request file has the following format:

```
COUNT = 0
Z = 64a9f6aeaced9959344765dc8810aaa8ac826ad6afb1a959b628de5f
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 = 64a9f6aeaced9959344765dc8810aaa8ac826ad6afb1a959b628de5f
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 17.8
# 'SSH' information for ABC
# SHA sizes tested: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
# Generated on Wed Sep 16 09:48:24 2015
```

For each option, CAVS generates two set of 10 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, is 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. 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. 3-key TDES CBC has a block size of 64 bits and a key length of 192 bits. The AES ciphers have a 128 bit IV length and a 128, 192, or 256 bit encryption key length. An example for TDES-CBC:

```
[SHA-1]
[shared secret length = 2048]
[IV length = 128]
[encryption key length = 192]
```

Each trial in the request file has the following format:

```
COUNT = 0
K =
0000010100a00096ffd6c123e3655baa72802594edbc32aaa6daded0b429fa616c3b6bc077d1b6
03b7e66d78b57e1dd7a1ba9bfb02bf338b0d40883856d09245380db0b513ef4e837822bbdeee3c
b3a05140be1405d383bab65ca7e22ac09350e2dcff21b985419f19b6182c28c324d2cb0e31cb6b
0f99fe31b43feb4182fa2a321715db12707f28a8c72843bd204abad139c087570eb526367ef420
e50169ad98e8c38e7780a6030fa4387736534454a2cb9a07f711989ce91ea1ee052ed559cae590
d1f26e336b9341563c9b37cde17ca8dcdc9213b78480e06bf9b620e451a37b87f5bbda9b0c7e55
7566bfa458f693cacb5834c3eae68c7c9190c10f53414b1a9d159a
H = cb77c99c3ac835010753fcfb02073b83d4aa5aed
session_id = cb77c99c3ac835010753fcfb02073b83d4aa5aed
```

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 =
0000010100a00096ffd6c123e3655baa72802594edbc32aaa6daded0b429fa616c3b6bc077d1b6
03b7e66d78b57e1dd7a1ba9bfb02bf338b0d40883856d09245380db0b513ef4e837822bbdeee3c
b3a05140be1405d383bab65ca7e22ac09350e2dcff21b985419f19b6182c28c324d2cb0e31cb6b
0f99fe31b43feb4182fa2a321715db12707f28a8c72843bd204abad139c087570eb526367ef420
e50169ad98e8c38e7780a6030fa4387736534454a2cb9a07f711989ce91ea1ee052ed559cae590
d1f26e336b9341563c9b37cde17ca8dcdc9213b78480e06bf9b620e451a37b87f5bbda9b0c7e55
7566bfa458f693cacb5834c3eae68c7c9190c10f53414b1a9d159a
H = cb77c99c3ac835010753fcfb02073b83d4aa5aed
session_id = cb77c99c3ac835010753fcfb02073b83d4aa5aed
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 AES key length options tested on line three, the KDR key lengths tested on line four, and the date and time the file was generated on line five. Two sample SRTP file headers are shown below.

The first shows if all KDRs are tested:

```
# CAVS 18.0
# 'SRTP' information for abc
# AES key lengths tested: AES-128, AES-192, AES-256
# # KDR Key Lengths tested: KDR - All possible values tested
# Generated on Tue Mar 03 16:30:15 2015
```

```
The second shows if select KDRs are tested.  In this case, KDR=0 and KDR=2^1
is tested:

# CAVS 18.0
# 'SRTP' information for test1
# AES key lengths tested: AES-128
# KDR Key Lengths tested: KDR_0, KDR_2^1
# Generated on Tue Mar 03 16:30:51 2015
```

If all KDRs are being tested, CAVS generates a set of 100 trials for each AES key length supported. If only select KDRs are being tested, CAVS generates a set of 100 trials for each combination of AES key length and KDR supported.

The trials begin after the AES key length and the KDR headers. Some examples of header combinations:

```
[AES-256]
[KDR:All possible values]

[AES-128]
[KDR:KDR_0]

[AES-192]
[KDR:KDR_2^1]
```

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 10 trials for SNMP. Each set of 10 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 10 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.