Society of Cable Telecommunications Engineers

ENGINEERING COMMITTEE
Digital Video Subcommittee

AMERICAN NATIONAL STANDARD

ANIS/SCTE 201 2013

Open Media Security (OMS) Root Key Derivation Profiles and Test Vectors

NOTICE

Table of Contents

1 Abstract 1
  1.1 Background 1
  1.2 Introduction 1
  1.3 Normative References 2
  1.4 Informative References 3
  1.5 Definitions 4
2 Functional Diagram 6
3 Base Requirements 8
4 Additional Requirements 9
  4.1 Preliminary SCK Manipulation Function 9
    4.1.1 Triple-DES 9
    4.1.2 AES Encrypt 9
    4.1.3 AES Decrypt 10
  4.2 Vendor Separation Function 10
    4.2.1 Triple-DES 11
    4.2.2 AES Encrypt 11
    4.2.3 AES Decrypt 12
  4.3 Final Root Key Derivation Function 12
    4.3.1 Triple-DES 12
    4.3.2 AES Encrypt 13
    4.3.3 AES Decrypt 13
  4.4 Module Key Derivation Function 14
    4.4.1 Triple-DES 14
    4.4.2 AES Encrypt 15
    4.4.3 AES Decrypt 15
5 Profiles 16
  5.1 Summary 16
  5.2 Profile 0 – Base Profile 17
  5.3 Profile 1 – Triple DES Profile 17
  5.4 Profile 1A – Triple DES Profile with Module Key Derivation 17
  5.5 Profile 2 – AES Profile 17
  5.6 Profile 2A – AES Encrypt Profile with Module Key Derivation 18
  5.7 Profile 2B – AES Decrypt Profile with Module Key Derivation 18
6 Test Vectors 19
  6.1 Root Key Derivation Test Vectors 19
    6.1.1 Profile 0 Operation 19
    6.1.2 Profile 1 Operation 19
    6.1.3 Profile 1A Operation 20
    6.1.4 Profile 2 Operation 21
    6.1.5 Profile 2A Operation 22
    6.1.6 Profile 2B Operation 24
  6.2 Content Descrambling (CW) Vectors 25
    6.2.1 DVB CSA2 Operation 25
    6.2.2 AES Vector 25
Appendix A Test Vectors Python Script 27

List of Tables

Table 1 Key Ladder / Content Descrambling Algorithms 16

List of Figures

Figure 1 OMS Key Ladder Functional Diagram 6
Figure 2 Root Key Derivation Functionality 7

1 Abstract

This cryptographic key ladder standard defines a set of key ladder profiles, additional requirements and test vectors for a key ladder implementation.

1.1 Background

This standard is an extension of the ETSI TS 103 162 [1] standard for a key ladder, by further defining certain aspects and providing test vectors to enable implementers to verify certain aspects of an implementation.

The use of a standard key ladder is part of enabling any television receiving device to receive scrambled television content from any television distribution network, independent of the network conditional access security system in use.

However, use of ETSI TS 103 162 [1], described below as Profile 0, is discouraged as it allows use of undisclosed algorithms and therefore undisclosed and unknown intellectual property. This standard specifies certain processes which are both necessary for interoperability and not specified in the ETSI standard.

1.2 Introduction

The key ladder is a standard for enabling and securing the delivery of content descrambling keys from a source device to a sink device. The key ladder derivation is described in this standard, and is a component of a larger system, referred to in this standard as the Open Media Security (OMS).

The basis of the key ladder standard is a three-step key ladder and challenge-response authentication scheme in which the base key derivation inputs are protected within the one-time programmable memory (OTP) of the sink device's hardware (e.g. chipset). The key ladder is used primarily for the delivery of content descrambling keys while the challenge-response mechanism is used for checking the integrity and authenticity of sink devices as well as messages arriving from a source.

The key ladder standard is designed to support dynamic substitution and replacement of either sink or source device in a manner that maintains the security and integrity of the underlying content distribution network. The standard enables the portability of sink devices between content distribution networks by permitting the field upgradeability of sink devices to work with previously unknown source devices. The standard also enhances the capability of networks to upgrade their source devices without disrupting the capabilities of already fielded sink devices.

The source device is expected to be a key management system such as a traditional CAS or DRM solution deployed by a content distribution network, and the sink device is expected to be a secure content consumption device such as a STB or television, this standard is not limited to only supporting these particular types of devices.

The root key derivation function yields a different set of keys for different Vendor_ID values, yielding a system where several different conditional access systems can simultaneously operate separately, securely. Similarly, where Module_ID is used, different values of Module_ID yield different keys, which are used for, e.g., DRM functions.

This standard does not specify how content arrives to the OMS sink device descrambler, only that the OMS sink device's descrambler must recognize the scrambling algorithm utilized by the content's network distribution system.

This standard does not specify compliance and robustness rules for chipset hardware nor interoperability or certification requirements. Such rules are beyond the scope of this standard and are expected to be the responsibility of an industry-wide licensing authority (ILA).

It is recognized that effective and safe implementation and deployment of content security systems based on the mechanisms described in the present document will require a complete security infrastructure that can deal with business, security, intellectual property, documentation and trusted information distribution issues. The description of such an infrastructure and the organizations which will administer it (i.e. an ILA) is outside of the scope of the present document.

As OMS is expected to be implemented in the chipset hardware of OMS sink devices, a universal separable security standard would also expect that the OMS sink device's hardware would implement all standardized descrambling algorithms that it might ever encounter. To ensure universal portability of OMS sink device hardware between networks, a finite set of descrambling algorithms is implemented in these devices.

1.3 Normative References

The following documents contain provisions, which, through reference in this text, constitute provisions of the standard. At the time of Subcommittee approval, the editions indicated were valid. All standards are subject to revision; and while parties to any agreement based on this standard are encouraged to investigate the possibility of applying the most recent editions of the documents listed below, they are reminded that newer editions of those documents may not be compatible with the referenced version.

[1] ETSI TS 103 162 v1.1.1 Access, Terminals, Transmission and Multiplexing (ATTM); Integrated Broadband Cable and Television Networks; K-LAD Functional Specification, October 2010.

[2] ANSI X9.52 (TDES) Triple-DES Block Cipher Triple Data Encryption Algorithm Modes Of Operation, X9.52 – 1998, Accredited Standards Committee X9, American National Standards Institute, July 27, 1998.

[3] FIPS-197 (AES) Specification for the Advanced Encryption Standard Federal Information Processing Standards (FIPS) Publication 197, November 26, 2001.

1.4 Informative References

[4] GY/T 255-2012 Technical specification of downloadable conditional access system, China Communications Standards Association, March 16, 2013, see http://www.ptsn.net.cn/standard/std_query/show-gy-333-1.htm

[5] ETSI TS 100 289 v1.1.1 Digital Video Broadcasting (DVB); Support for use of the DVB Scrambling Algorithm version 3 within digital broadcasting systems, September 2011.

1.5 Definitions

- *Chip ID* (**Chip-ID**) is an 8-byte public identifier of the OMS sink device chipset, including elements indicating the manufacturer and model as well as a globally unique identifier for the chipset instance within that model.
- *SCK* (**SCK**) is the secret chipset key which is unique to each OMS sink device chipset. Must be at least 16-bytes.
- ESCK (ESCK) is the obfuscated secret chipset key which is the value physically stored in the chipset's OTP. Must be at least as large as the SCK. The ESCK would be typically uneditable and unreadable after manufacture.
- Secret Mask Key is an embedded secret value, created by the manufacturer and physically stored in the chipset. Values of Secret Mask Key can be common among several different component versions (for example, when several different chip models share a base die). Must be at least 16-bytes. Further constraints are outside the scope of this document.
- *Key Ladder Root Key,* or *Root Key* (**K3**) is the 16-byte secret key used at the root of the key ladder to decrypt **K2**. In chipsets that implement an extended key ladder with *n* levels, the root key at the highest level of the key ladder will be denoted by **Kn.**
- *Control Word* (**CW**) is the key used to descramble the video, either 8 or 16 bytes depending on the key length for the chosen descrambling algorithm.
- Open Media Security (**OMS**) is any system that uses the key ladder described in this standard.
- *OMS Key 1* (**K1**) is a 16-byte key used to decrypt the **CW**.
- *OMS Key 2* (**K2**) is a 16-byte key used to decrypt **K1**.
- *Authentication key* (**A**) is a 16-byte key derived from **K2** that is used by the challenge-response mechanism. **A** can be used either to authenticate the sink device through a traditional challenge-response, or used by the sink device to authenticate messages from the source device by deriving a key for a CBC-MAC or similar symmetric message authentication algorithm.
- *Vendor_ID* is a value that is used to identify CA vendors, MSOs, and other entities using an OMS chipset. The size of Vendor_ID is determined by the profile in use.
- *Module_ID* is an 8-bit value that is used, in certain profiles, to generate additional, cryptographically linked, keys related to K3.

- *PID* is a Packet ID of a component elementary stream within a program carried in an MPEG-2 transport stream.
- ***Ek(Y)*** is used to denote the data Y encrypted with key K.
- ***Triple-DES* or *T-DES*** means the "Triple DES (TDES)" cipher as described in TS 103 162, Section 6.1.3, namely it "means two-key Triple DES. If the two keys are A and B, then the decryption function should be $\mathbf{D_A(E_B(D_A(x)))}$. When decrypting more than 64 bits (the block size of TDES), the cipher shall be used in ECB mode. The key parity bits shall be ignored."

2 Functional Diagram

Figure 1 shows an overview of the OMS key ladder. Figure 1 does not represent actual hardware architecture.

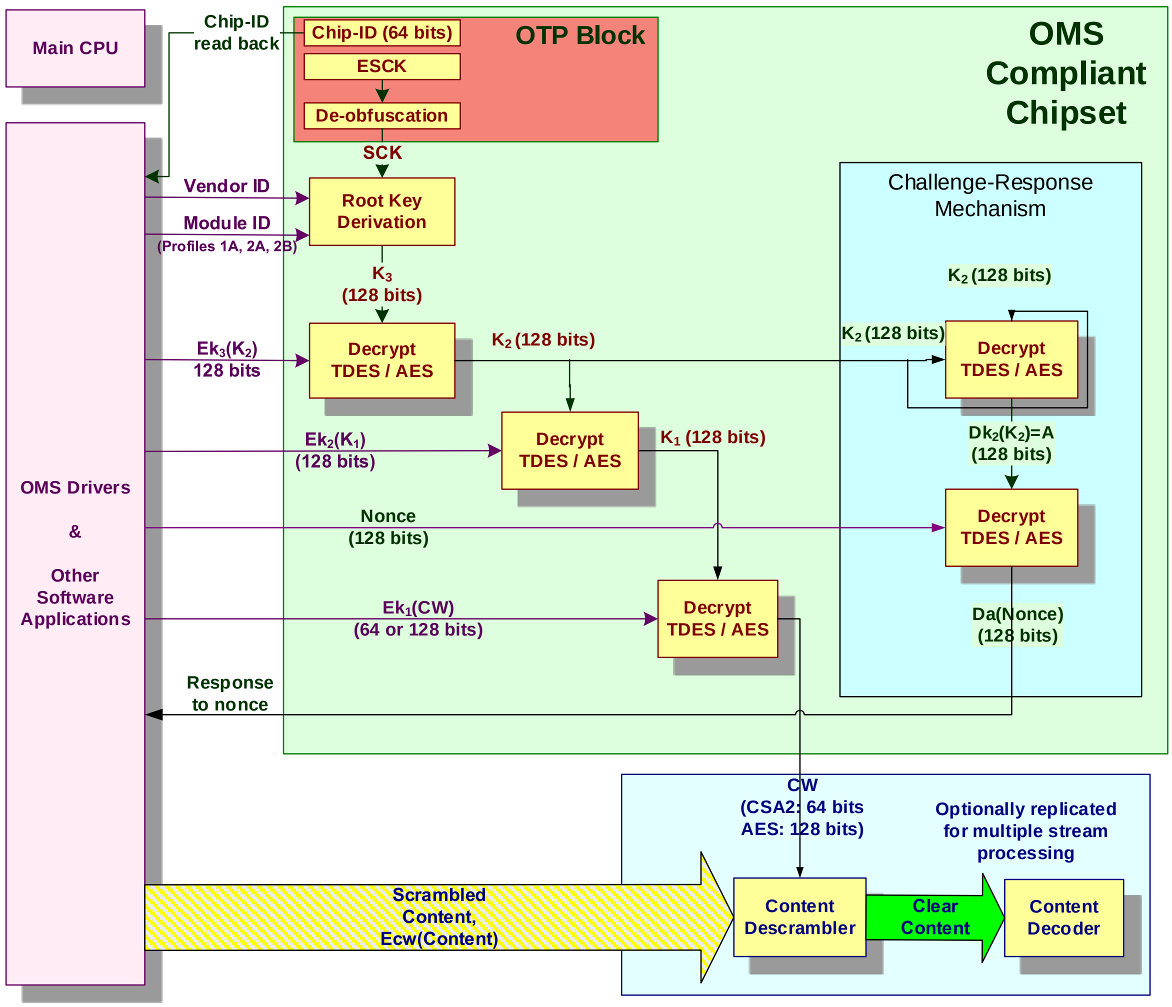

Figure 1 OMS Key Ladder Functional Diagram

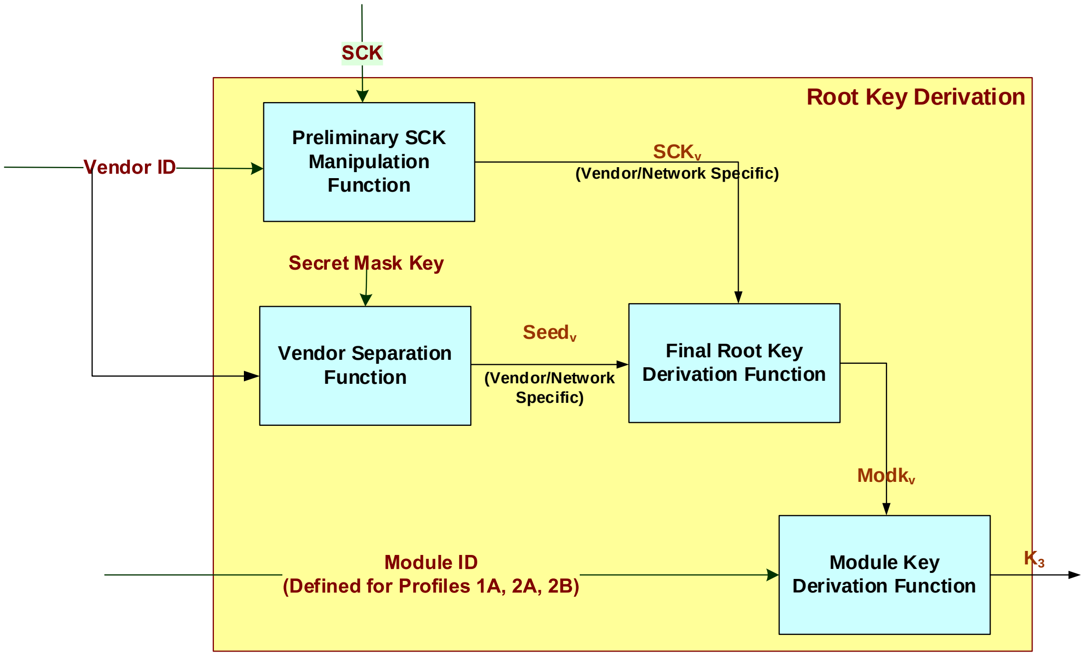

Figure 2 Root Key Derivation

3 Base Requirements

All devices shall comply with the normative requirements of ETSI TS 103 162 [1], including the requirement that devices include both DVB CSA2 and AES-128 bit CBC descrambling capability, see ETSI TS 103 162 [1] Section 4.2.

It is expected that future versions of this standard may include DVB CSA3 as a descrambling algorithm, see ETSI TS 103 162 [1] Section 4.2 and ETSI TR 100 289 [5].

4 Additional Requirements

This standard specifies certain optional features described below. Implementation of certain of these optional features is required for certain profiles of this standard, as described in Section 5, Profiles.

4.1 Preliminary SCK Manipulation Function

The following sections contain multiple definitions of the Preliminary SCK Manipulation Function. The choice of which definition of the Preliminary SCK Manipulation Function is covered by the profiles described in Section 5.

4.1.1 Triple-DES

For certain profiles, the Preliminary SCK Manipulation uses the Triple-DES cipher described in ANSI X9.52 [2] and ETSI TS 103 162 [1] Section 6.1.3, operating in two-key Triple DES mode. When operating on more than 64 bits (the block size of TDES), the Triple-DES cipher shall be used in ECB mode. The key parity bits shall be ignored.

The operation uses the 16-byte SCK for the key, and 0x01000000 0000VVVV 02000000 0000VVVV (where VVVV is the 16-bit Vendor_ID) for the data

The Preliminary SCK Manipulation function, used to generate SCK_V, shall be:

$$PSCK = D_A\left(E_B\left(D_A(X)\right)\right)$$

Where:

- $D_A(data)$ means DES decryption of $data$ with key A in ECB mode with parity bits ignored
- $E_B(data)$ means DES encryption of $data$ with key B in ECB mode with parity bits ignored
- A shall be most-significant 64 bits of the 16-byte SCK.
- B shall be the least-significant 64 bits of the 16-byte SCK.
- X shall be 0x0100 0000 0000 vvvv 0200 0000 0000 vvvv, where 'vvvv' is the 16-bit Vendor_ID.

4.1.2 AES Encrypt

For certain profiles, the Preliminary SCK Manipulation uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode.

The operation uses the 16-byte SCK for the key, and fourteen bytes of 0x00, followed by the 16-bit Vendor_ID which comprise the two least significant bytes as the data (e.g., 0x00000000 00000000 00000000 0000 VVVV)

The Preliminary SCK Manipulation function, for profiles that use AES Encrypt to generate SCK_V, shall be:

$$PSCK = E_A(X)$$

Where:

- $E_A(data)$ means AES encryption of $data$ with key A in ECB mode.
- A shall be the 16-byte SCK.
- X shall be 0x0000 0000 0000 0000 0000 0000 0000 vvvv, where 'vvvv' is the 16-bit Vendor_ID.

4.1.3 AES Decrypt

For certain profiles, the Preliminary SCK Manipulation uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode.

The operation uses the 16-byte SCK for the key, and fourteen bytes of 0x00, followed by the 16-bit Vendor_ID which comprise the two least significant bytes as the data (e.g., 0x00000000 00000000 00000000 0000 VVVV)

The Preliminary SCK Manipulation function, for profiles that use AES Decrypt to generate SCK_V, shall be:

$$PSCK = D_A(X)$$

Where:

- $D_A(data)$ means AES decryption of $data$ with key A in ECB mode.
- A shall be the 16-byte SCK.
- X shall be 0x0000 0000 0000 0000 0000 0000 0000 vvvv, where 'vvvv' is the 16-bit Vendor_ID.

4.2 Vendor Separation Function

The following sections contain multiple definitions of the Vendor Separation Function. The choice of which definition of the Vendor Separation Function is covered by the profiles described in Section 5.

4.2.1 Triple-DES

For certain profiles, the Vendor Separation Function uses the Triple-DES cipher described in ANSI X9.52 [2] and ETSI TS 103 162 [1] Section 6.1.3 operating in two-key Triple DES mode. When operating on more than 64 bits (the block size of TDES), the Triple-DES cipher shall be used in ECB mode. The key parity bits shall be ignored.

The operation uses the 16-byte Secret Mask Key for the key, and 0x01000000 0000VVVV 02000000 0000VVVV (where VVVV is the 16-bit Vendor_ID) for the data.

The Vendor Separation Function, used to generate Seed_V, shall be:

$$VSF = D_A\left(E_B\left(D_A(X)\right)\right)$$

Where:

- $D_A(data)$ means DES decryption of $data$ with key A in ECB mode with parity bits ignored
- $E_B(data)$ means DES encryption of $data$ with key B in ECB mode with parity bits ignored
- A shall be most-significant 64 bits of the 16-byte Secret Mask Key.
- B shall be the least-significant 64 bits of the 16-byte Secret Mask Key.
- X shall be 0x0100 0000 0000 vvvv 0200 0000 0000 vvvv, where 'vvvv' is the 16-bit Vendor_ID.

4.2.2 AES Encrypt

For certain profiles, the Vendor Separation Function uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode.

The operation uses the 16-byte Secret Mask Key for the key, and fourteen bytes of 0x00, followed by the 16-bit Vendor_ID which comprise the two least significant bytes as the data (e.g., 0x00000000 00000000 00000000 0000 VVVV).

The Vendor Separation Function, for profiles that use AES Encrypt to generate Seed_V, shall be:

$$VSF = E_A(X)$$

Where:

- $E_A(data)$ means AES encryption of $data$ with key A in ECB mode
- A shall be the 16-byte Secret Mask Key.

- X shall be 0x0000 0000 0000 0000 0000 0000 0000 vvvv, where 'vvvv' is the 16-bit Vendor_ID.

4.2.3 AES Decrypt

For certain profiles, the Vendor Separation Function uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode.

The operation uses the 16-byte Secret Mask Key for the key, and fourteen bytes of 0x00, followed by the 16-bit Vendor_ID which comprise the two least significant bytes as the data (e.g., 0x00000000 00000000 00000000 0000 VVVV).

The Vendor Separation Function, for profiles that use AES Decrypt to generate $Seed_V$, shall be:

$$VSF = D_A(X)$$

Where:

- $D_A(data)$ means AES decryption of $data$ with key A in ECB mode
- A shall be the 16-byte Secret Mask Key.
- X shall be 0x0000 0000 0000 0000 0000 0000 0000 vvvv, where 'vvvv' is the 16-bit Vendor_ID.

4.3 Final Root Key Derivation Function

The following sections contain multiple definitions of the Final Root Key Derivation Function. The choice of which definition of the Final Root Key Derivation Function is covered by the profiles described in Section 5.

4.3.1 Triple-DES

For certain profiles, the Final Root Key Derivation Function uses the Triple-DES cipher described in ANSI X9.52 [2] and ETSI TS 103 162 [1] Section 6.1.3 operating in two-key Triple DES mode, with a final XOR operation. When operating on more than 64 bits (the block size of TDES), the Triple-DES cipher shall be used in ECB mode. The key parity bits shall be ignored.

The operation uses the 16-byte SCK_V for the key, and the data for the operation is $Seed_V$.

The Final Root Key Derivation Function, used to generate $Modk_v$, shall be:

$$FRKD = \left(D_A\left(E_B\left(D_A(X)\right)\right)\right) \oplus \mathrm{X}$$

Where:

- $D_A(data)$ means DES decryption of $data$ with key A in ECB mode with parity bits ignored
- $E_B(data)$ means DES encryption of $data$ with key B in ECB mode with parity bits ignored
- A shall be most-significant 64 bits of the 16-byte SCK_V.
- B shall be the least-significant 64 bits of the 16-byte SCK_V.
- X shall be $Seed_V$.
- $\oplus$ means the bitwise exclusive-or operation

4.3.2 AES Encrypt

For certain profiles, the Final Root Key Derivation Function uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode, with a final XOR operation.

The operation uses the 16-byte SCK_V for the key, and $Seed_V$ for the data, with the result XORed with $Seed_v$

The Final Root Key Derivation Function, for profiles that use AES Encrypt to generate $Modk_v$, shall be:

$$FRKD = \left(E_A(X)\right) \oplus \mathrm{X}$$

Where:

- $E(data)$ means AES encryption of $data$ with key A in ECB mode
- A shall be the 16-byte SCK_V.
- X shall be $Seed_V$.
- $\oplus$ means the bitwise exclusive-or operation

4.3.3 AES Decrypt

For certain profiles, the Final Root Key Derivation Function uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode, with a final XOR operation.

The operation uses the 16-byte SCK_V for the key, and $Seed_V$ for the data, with the result XORed with $Seed_v$

The Final Root Key Derivation Function, for profiles that use AES Decrypt to generate $Modk_v$, shall be:

$$FRKD = \left(D_A(X)\right) \oplus \mathrm{X}$$

Where:

- $D_A(data)$ means AES decryption of $data$ with key A in ECB mode
- A shall be the 16-byte SCK_V.
- X shall be $Seed_V$.
- $\oplus$ means the bitwise exclusive-or operation

4.4 Module Key Derivation Function

The following sections contain multiple definitions of the Additional Root Key Derivation Function. Certain profiles described in Section 5 include an Module Key Derivation. The choice of which definition of the Module Key Derivation Function, if any, is covered by the profiles described in Section 5.

4.4.1 Triple-DES

For certain profiles, the Additional Root Key Derivation Function uses the Triple-DES cipher described in ANSI X9.52 [2] and ETSI TS 103 162 [1] Section 6.1.3 operating in two-key Triple DES mode. When operating on more than 64 bits (the block size of TDES), the Triple-DES cipher shall be used in ECB mode. The key parity bits shall be ignored.

The operation uses the 16-byte $Modk_v$ for the key, and 0x01000000 000000MM 0200000000 000000MM (where MM is the 8-bit Module_ID) for the data.

The Module Key Derivation Function, used to generate K_3, shall be:

$$MKD = D_A\left(E_B\left(D_A(X)\right)\right)$$

Where:

- $D_A(data)$ means DES decryption of $data$ with key A in ECB mode with parity bits ignored
- $E_B(data)$ means DES encryption of $data$ with key B in ECB mode with parity bits ignored
- A shall be most-significant 64 bits of the 16-byte $Modk_v$.
- B shall be the least-significant 64 bits of the 16-byte $Modk_v$.
- X shall be 0x01000000 000000MM 0200000000 000000MM (where MM is the 8-bit Module_ID).

4.4.2 AES Encrypt

For certain profiles, the Module Key Derivation Function uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode.

The operation uses the 16-byte $Modk_v$ for the key, and Module_ID for the data.

The Module Key Derivation Function, for profiles that use AES Encrypt to generate K_3, shall be:

$$MKD = E_A(X)$$

Where:

- $E_A(data)$ means AES encryption of $data$ with key A in ECB mode
- A shall be the 16-byte $Modk_v$.
- X shall be the 0x0000 0000 0000 0000 0000 0000 0000 00MM (where MM is the 8-bit Module_ID).

4.4.3 AES Decrypt

For certain profiles, the Module Key Derivation Function uses the Advanced Encryption Standard described in FIPS-197 [3] operating in AES-128 ECB mode.

The operation uses the 16-byte $Modk_v$ for the key, and Module_ID for the data.

The Module Key Derivation Function, for profiles that use AES Decrypt to generate K_3, shall be:

$$MKD = D_A(X)$$

Where:

- $D_A(data)$ means AES decryption of $data$ with key A in ECB mode
- A shall be the 16-byte $Modk_v$.
- X shall be the 0x0000 0000 0000 0000 0000 0000 0000 00MM (where MM is the 8-bit Module_ID).

5 Profiles

5.1 Summary

Table 1 below shows the base profile and the additional profiles in this standard.

Note: Particular implementations may support more than one profile.

Profile	Preliminary SCK Manipulation Function	Vendor Separation Function	Final Root Key Derivation Function	Module Key Derivation Function
Profile 0 – Base Profile	Not Explicitly Defined	Not Explicitly Defined	Not Explicitly Defined	Not Explicitly Defined
Profile 1 – Triple DES Profile	T-DES	T-DES	T-DES	n/a
Profile 1a – Triple DES Profile with Module Key Derivation	T-DES	T-DES	T-DES	T-DES
Profile 2 – AES Profile	AES Encrypt	AES Encrypt	AES Encrypt	n/a
Profile 2a – AES Encrypt Profile with Module Key Derivation	AES Encrypt	AES Encrypt	AES Encrypt	AES Encrypt
Profile 2b – AES Decrypt Profile with Module Key Derivation	AES Decrypt	AES Decrypt	AES Decrypt	AES Decrypt

Table 1 - Profiles

5.2 Profile 0 – Base Profile

Devices that comply with Profile 0, the Base Profile, shall comply with the normative requirements in Section 3, and may comply with other aspects of this standard. Profile 0 is a 'base' profile, and devices that comply with the other profiles comply with Profile 0 by definition.

5.3 Profile 1 – Triple DES Profile

Devices that comply with Profile 1, the Triple DES Profile, shall comply with the normative requirement of Profile 0, see section 5.1, and shall comply with the normative requirements of the following:

- Section 4.1.1, requiring use of Triple-DES for the Preliminary SCK Manipulation Function;
- Section 4.2.1, requiring use of Triple-DES for the Vendor Separation Function; and
- Section 4.3.1, requiring use of Triple-DES for the Final Root Key Derivation Function.
- Vendor_ID shall be 16 bits.
- K_3 is defined to be $Modk_v$.

5.4 Profile 1A – Triple DES Profile with Module Key Derivation

Devices that comply with Profile 1A, the Triple DES Profile with Module Key Derivation, shall comply with the normative requirements of Profile 1 (see section 5.3) to the extent that Profile 1 is not in conflict with the following, and shall comply with the normative requirements set forth below.

- Section 4.4.1, requiring the use of Triple-DES for the Module Key Derivation Function; K_3 is calculated (not defined as in Profile 1)
- Module_ID shall be 8 bits.

5.5 Profile 2 – AES Profile

Devices that comply with Profile 2, the AES Profile, shall comply with the normative requirement of Profile 0, see section 5.1, and shall comply with the normative requirements of the following:

- Section 4.1.2, requiring use of AES Encrypt for the Preliminary SCK Manipulation Function;
- Section 4.2.2, requiring use of AES Encrypt for the Vendor Separation Function; and

- Section 4.3.2, requiring use of AES Encrypt for the Final Root Key Derivation Function.
- Vendor_ID shall be 16 bits.
- K_3 is defined to be $Modk_v$.

5.6 Profile 2A – AES Encrypt Profile with Module Key Derivation

Devices that comply with Profile 2A, the AES Encrypt Profile with Module Key Derivation, shall comply with the normative requirements of Profile 2 (see section 5.5) to the extent that Profile 2 is not in conflict with the following, and shall comply with the normative requirements set forth below:

- Section 4.4.2, requiring the use of AES Encrypt for the Module Key Derivation Function; K_3 is calculated (not defined as in Profile 2).
- Module_ID shall be 8 bits.

5.7 Profile 2B – AES Decrypt Profile with Module Key Derivation

Devices that comply with Profile 2B, the AES Decrypt Profile with Module Key Derivation, shall comply with the normative requirements of Profile 2 (see section 5.5) to the extent that Profile 2 is not in conflict with the following, and shall comply with the normative requirements set forth below:

- Section 4.1.3, requiring use of AES Decrypt for the Preliminary SCK Manipulation Function;
- Section 4.2.3, requiring use of AES Decrypt for the Vendor Separation Function; and
- Section 4.3.3, requiring use of AES Decrypt for the Final Root Key Derivation Function.
- Section 4.4.3, requiring the use of AES Decrypt for the Module Key Derivation Function; K_3 is calculated (not defined as in Profile 2).
- Module_ID shall be 8 bits.

6 Test Vectors

6.1 Root Key Derivation Test Vectors

When correctly operating in each profile below, calculations on the input values given will yield the output values given. Implementers should verify that implementations correctly yield these values.

Note that the test vectors for each profile have cases that correspond to the various implementations required by ETSI TS 103 162 [1]: $E_{k_1}(CW)$ may be 64 or 128 bits; CW may be 64 (for CSA descrambling) or 128 (for AES descrambling) bits; the K_2, K_1, Challenge-Response and CW computations may use TDES or AES.

6.1.1 Profile 0 Operation

No test vectors for Profile 0 Root Key Derivation are provided, as Profile 0 is not explicitly defined.

6.1.2 Profile 1 Operation

Field	Value (hex msb)
Root Key Computation	
SCK	77 65 6C 63 6F 6D 65 74 6F 6D 79 70 61 72 74 79
Mask Key	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Vendor ID	2A 42
SCKv	8A 40 B1 FE 49 23 1C 52 56 7D 23 6B 0D AF CA AF
Seedv	35 FD 89 47 57 B4 C0 45 34 80 F6 7E A3 1D DB 8F
K3	84 D5 EA 9E E5 75 27 30 93 64 2B D8 87 66 9F CF
K1 Computation (AES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	7C 82 1D 5F 6A F8 26 37 2D 39 E5 0D 88 4A D3 60
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	45 2A C2 C1 C9 44 61 CB 63 45 64 78 71 67 33 54
CW Computation (Vector 1) (AES/AES)	
ECW	D5 38 E6 01 74 41 48 8D D2 12 A4 F1 69 7E F0 52
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (AES/CSA)	
ECW	BC 96 F2 33 CA 34 6B 29 51 FF 52 50 7F 76 BA 6B
CW	BC FB B2 69 13 BA BE 8B 00 00 00 00 00 00 00 00
CW Computation (Vector 3) (AES/CSA)	
ECW	24 0E 04 47 C0 26 A5 FA 40 65 C9 FC A8 F6 C1 A0
CW	68 E1 DA 5B 24 AD 86 1F 00 00 00 00 00 00 00 00

Challenge-Response Computation (AES)	
A	59 1D F7 FD 66 D5 78 85 35 15 FC 6E 7B 7A 37 B0
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	E8 39 FB 00 8F 4B 54 5E F7 79 F9 86 C0 74 CE 9A
K1 Computation (TDES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	C1 6B F4 B2 04 33 98 7D 00 1E 68 DF 54 96 0F 8F
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	50 BD EB F4 64 8C 13 02 4B 33 23 11 F4 F6 AF DE
CW Computation (Vector 1) (TDES/AES)	
ECW	B3 A9 5B 27 DC 86 7E 38 C9 A8 F8 D0 2E F6 26 55
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (TDES/CSA)	
ECW	3A 77 C8 80 A4 2A F2 BB
CW	BC FB B2 69 13 BA BE 8B
CW Computation (Vector 3) (TDES/CSA)	
ECW	B3 A9 5B 27 DC 86 7E 38
CW	68 E1 DA 5B 24 AD 86 1F
Challenge-Response Computation (TDES)	
A	E3 A2 C5 14 7A 45 71 D7 1C 7E 8F 47 52 54 26 BF
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	4C E1 50 53 BE 7C 92 81 1F D3 41 45 30 11 C9 8A

6.1.3 Profile 1A Operation

Field	Value (hex msb)
Root Key Computation	
SCK	77 65 6C 63 6F 6D 65 74 6F 6D 79 70 61 72 74 79
Mask Key	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Vendor ID	2A 42
SCKv	8A 40 B1 FE 49 23 1C 52 56 7D 23 6B 0D AF CA AF
Seedv	35 FD 89 47 57 B4 C0 45 34 80 F6 7E A3 1D DB 8F
Module ID	A5
K3	FE E7 0C DE A9 2D C5 1E D9 82 4A F1 4B 8F A2 D3
K1 Computation (AES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	10 5D D1 D8 42 C5 AD D1 F5 FA 1C F1 F3 63 CC 44
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	A7 41 46 30 49 C9 1D C1 DA 1C 86 C8 01 D2 63 17
CW Computation (Vector 1) (AES/AES)	
ECW	AC 43 D0 60 78 68 9D 12 46 0B 53 0A FF 3E 09 E4

CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (AES/CSA)	
ECW	F6 A0 35 EC 73 8A E1 65 0D 88 7D 34 38 A4 48 1A
CW	BC FB B2 69 13 BA BE 8B 00 00 00 00 00 00 00 00
CW Computation (Vector 3) (AES/CSA)	
ECW	19 D7 DA 3B 84 9E B3 11 11 06 87 AA 39 6C 34 BA
CW	68 E1 DA 5B 24 AD 86 1F 00 00 00 00 00 00 00 00
Challenge-Response Computation (AES)	
A	8E 18 23 A6 2C 06 B0 C5 50 3F FB 34 48 DB BE EF
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	49 55 02 8B DA 07 F9 6D B8 EE B9 B3 C4 9E B1 F7
K1 Computation (TDES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	18 C4 F9 69 C5 DC A1 C4 96 D5 F9 D5 08 82 5A CC
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	28 4D 0F 68 37 EA 32 94 49 F1 43 80 5A 1D 50 87
CW Computation (Vector 1) (TDES/AES)	
ECW	EA 09 F2 7B E2 76 99 44 F3 80 E1 AF 89 3B 85 34
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (TDES/CSA)	
ECW	6A CC 69 8E 4F B0 55 33
CW	BC FB B2 69 13 BA BE 8B
CW Computation (Vector 3) (TDES/CSA)	
ECW	EA 09 F2 7B E2 76 99 44
CW	68 E1 DA 5B 24 AD 86 1F
Challenge-Response Computation (TDES)	
A	DA C3 8D 1C BA C1 44 AA 15 D5 A2 D2 31 78 A4 E6
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	5A 51 4E 3E A0 9F F9 7B C0 24 3E D1 8C F1 E9 88

6.1.4 Profile 2 Operation

Field	Value (hex msb)
Root Key Computation	
SCK	77 65 6C 63 6F 6D 65 74 6F 6D 79 70 61 72 74 79
Mask Key	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Vendor ID	2A 42
SCKv	D4 54 0B A3 97 57 EF 40 E7 2E 03 8A 1F 2D 2C 88
Seedv	4D B0 F4 D5 A1 2E 3E 00 CC FD 9B C7 B7 3B 52 B7

K3	E3 91 61 63 F1 E4 E0 D7 75 3A CC 77 BE C6 6F 3B
K1 Computation (AES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	FA A3 62 9B 21 A6 7B 1B BB 85 BB C1 9E D5 F1 25
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	E7 D3 6B 20 28 41 64 46 CA 6E 04 C2 4A 51 F1 42
CW Computation (Vector 1) (AES)	
ECW	5F 5F 40 3A E2 60 CD 2B 6B 62 81 0D 18 F9 65 03
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (AES/CSA)	
ECW	B1 D0 02 F4 01 1F AD 40 37 9E 9E C7 1C 1A D3 9E
CW	BC FB B2 69 13 BA BE 8B 00 00 00 00 00 00 00 00
CW Computation (Vector 3) (AES/CSA)	
ECW	52 3E 58 99 FC 2C 48 E1 F3 FB 29 93 06 3C 4B 3A
CW	68 E1 DA 5B 24 AD 86 1F 00 00 00 00 00 00 00 00
Challenge-Response Computation (AES)	
A	CE A8 DF 37 21 E9 50 94 22 00 49 C1 DC 43 82 70
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	4F 92 6A 71 1F DB 61 06 9D A4 32 D3 1C 94 E9 47
K1 Computation (TDES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	B3 AD 35 00 41 CD 11 51 2D 67 E1 EE 8A 32 70 4E
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	65 FC 24 CB 22 15 26 A5 6A 23 7B AF B6 62 94 7F
CW Computation (Vector 1) (TDES/AES)	
ECW	F9 D0 6E FC 4F 1C BF 87 A3 DF ED 67 C1 7F 91 93
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (TDES/CSA)	
ECW	BE BF F8 D4 AB EF 7A 63
CW	BC FB B2 69 13 BA BE 8B
CW Computation (Vector 3) (TDES/CSA)	
ECW	F9 D0 6E FC 4F 1C BF 87
CW	68 E1 DA 5B 24 AD 86 1F
Challenge-Response Computation (TDES)	
A	6E DA E7 19 6E DC 62 E8 74 8C 27 3E B6 64 59 B3
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	EA 6A E6 3C C0 42 3B 5A 96 75 E7 65 C6 6A 34 ED

6.1.5 Profile 2A Operation

Field	Value (hex msb)

Root Key Computation	
SCK	77 65 6C 63 6F 6D 65 74 6F 6D 79 70 61 72 74 79
Mask Key	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Vendor ID	2A 42
SCKv	D4 54 0B A3 97 57 EF 40 E7 2E 03 8A 1F 2D 2C 88
Seedv	4D B0 F4 D5 A1 2E 3E 00 CC FD 9B C7 B7 3B 52 B7
Module ID	A5
K3	76 94 74 29 8E 9C FC E1 46 2D 9C EE 1F 08 A2 CE
K1 Computation (AES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	67 E5 1F 2B 4C 9A 66 34 21 BB 2E AF 39 6E 24 56
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	F7 68 04 70 B9 01 B0 50 5E D0 FE 30 C1 9F C3 11
CW Computation (Vector 1) (AES/AES)	
ECW	89 BB 35 F7 BA 73 BB 72 62 A3 FD 42 74 7E EE CB
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (AES/CSA)	
ECW	0B 03 50 87 50 49 7A FE 4B 74 D2 63 E9 FD 01 78
CW	BC FB B2 69 13 BA BE 8B 00 00 00 00 00 00 00 00
CW Computation (Vector 3) (AES/CSA)	
ECW	FA 65 1F D6 58 DC 88 5C 08 FB 1C CC 4D 85 FF CA
CW	68 E1 DA 5B 24 AD 86 1F 00 00 00 00 00 00 00 00
Challenge-Response Computation (AES)	
A	A4 12 81 1E 7C 7E 81 51 55 12 EC 21 78 92 B3 1A
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	A4 0F 00 30 9B 60 18 E6 B1 D1 93 80 B3 65 81 B6
K1 Computation (TDES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	78 4C 1C D1 90 DE 6E 9C 74 B2 82 AE 13 01 A9 05
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	77 9F 60 47 7B 7E ED B9 77 F0 7F 14 22 2D AC D2
CW Computation (Vector 1) (TDES/AES)	
ECW	35 53 DB 4B B1 5F 05 51 8F CC FD 2F 80 CD 0E D1
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (TDES/CSA)	
ECW	D4 F5 83 47 7E F1 E3 26
CW	BC FB B2 69 13 BA BE 8B
CW Computation (Vector 3) (TDES/CSA)	
ECW	35 53 DB 4B B1 5F 05 51
CW	68 E1 DA 5B 24 AD 86 1F
Challenge-Response Computation (TDES)	
A	BC 7F DC B5 24 37 3F 41 00 A8 7D 4D FD 80 50 68
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	30 E3 CE 1A 96 47 2D 98 30 4B 1C 90 B5 43 78 25

6.1.6 Profile 2B Operation

Field	Value (hex msb)
Root Key Computation	
SCK	77 65 6C 63 6F 6D 65 74 6F 6D 79 70 61 72 74 79
Mask Key	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Vendor ID	2A 42
SCKv	05 20 6E AB EC 5E 95 80 12 5A A4 D9 92 7F 75 4B
Seedv	DC 59 AE D9 71 01 5D A5 C3 AA 5B 6B 8D DE EA D3
Module ID	A5
K3	64 B4 FF 72 DF D2 3A 4C EA 8E 62 7A F9 D5 5C D0
K1 Computation (AES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	77 EE 6E AB E3 8A 97 86 CB 13 E4 97 1D 69 F0 6D
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	9D 0F 28 D8 5A B3 09 F7 5E 62 A0 D2 8B 36 86 76
CW Computation (Vector 1) (AES/AES)	
ECW	4D BE A3 E8 62 E9 99 0A 1C 5A 7A 9F 78 0C 9C C0
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (AES/CSA)	
ECW	0D 6F 1A 91 F8 F4 90 AD D3 91 E2 98 EE DA 75 A9
CW	BC FB B2 69 13 BA BE 8B 00 00 00 00 00 00 00 00
CW Computation (Vector 3) (AES/CSA)	
ECW	25 29 2B FF 8D 4B D9 BE AF 4F 86 32 32 1E 16 B4
CW	68 E1 DA 5B 24 AD 86 1F 00 00 00 00 00 00 00 00
Challenge-Response Computation (AES)	
A	96 B1 0A 16 45 4C 0A 14 DA 5F F7 88 50 E0 D9 52
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	60 B2 24 33 73 40 4B 65 3E A8 50 6D 68 FA 32 12
K1 Computation (TDES)	
EK2	20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
K2	53 7F 38 D7 12 9C 71 CC CA 82 13 E4 5F 1D EB 24
EK1	10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
K1	6D 89 B7 56 28 F8 E2 80 02 CA 73 28 55 19 5B 19
CW Computation (Vector 1) (TDES/AES)	
ECW	D9 37 7D 6E 51 9D 32 E3 4D B8 26 0F A5 37 60 EC
CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
CW Computation (Vector 2) (TDES/CSA)	
ECW	6E F7 2D 83 AF 3F C0 64
CW	BC FB B2 69 13 BA BE 8B
CW Computation (Vector 3) (TDES/CSA)	
ECW	D9 37 7D 6E 51 9D 32 E3
CW	68 E1 DA 5B 24 AD 86 1F

Challenge-Response Computation (TDES)	
A	28 99 7C C9 FC B1 2E 28 A5 51 9E B6 1F 02 FF E7
Nonce	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
D_a(Nonce)	17 66 8D 8B D3 D0 F0 55 F5 10 7B 5B 1E 1A FA 95

6.2 Content Descrambling (CW) Vectors

6.2.1 DVB CSA2 Operation

Where 64-bit CSA is used for content descrambling:

CW	68 E1 DA 5B 24 AD 86 1F
Scrambled Transport Stream Packet ($E_{CW}(Content)$)	47 00 64 90 DC FE D2 5B 4F A0 FF 9E 43 99 6C 4A 08 52 D5 80 EF 54 54 A0 9E F0 05 D8 F6 30 29 74 43 35 E4 3A B9 24 CC 61 51 D9 EC 4E C5 F9 C5 39 AE B7 BE 82 90 5E 14 38 82 17 4D D1 06 10 B5 01 6F 73 7C 10 67 D8 33 58 93 88 23 90 6B 84 98 C3 AE B4 43 AB 78 E0 56 DB E5 2A BD B4 27 DE 4E 39 AA AF CE AD 36 4F 7A 0C 21 17 AF 49 53 75 F0 0F 17 26 C4 A8 BD 7E BF B9 9D D6 8E 04 3D D9 DC 67 70 94 FC ED B8 19 8D 77 1B 81 31 43 07 4B 61 AE 2B BD 31 CF D2 D4 D1 9B 2A 91 EE 6C F5 11 24 75 50 E6 20 0A 4C D3 F7 26 5B E6 83 3F B0 34 28 81 C2 CD CD 79 AD 50 37 0F 16 89 13 3C
Clear Transport Stream Packet ($Content$)	47 00 64 10 23 BE 84 E1 6C D6 AE 52 90 49 F1 F1 BB E9 EB B3 A6 DB 3C 87 0C 3E 99 24 5E 0D 1C 06 B7 47 DE B3 12 4D C8 43 BB 8B A6 1F 03 5A 7D 09 38 25 1F 5D D4 CB FC 96 F5 45 3B 13 0D 89 0A 1C DB AE 32 20 9A 50 EE 40 78 36 FD 12 49 32 F6 9E 7D 49 DC AD 4F 14 F2 44 40 66 D0 6B C4 30 B7 32 3B A1 22 F6 22 91 9D E1 8B 1F DA B0 CA 99 02 B9 72 9D 49 2C 80 7E C5 99 D5 E9 80 B2 EA C9 CC 53 BF 67 D6 BF 14 D6 7E 2D DC 8E 66 83 EF 57 49 61 FF 69 8F 61 CD D1 1E 9D 9C 16 72 72 E6 1D F0 84 4F 4A 77 02 D7 E8 39 2C 53 CB C9 12 1E 33 74 9E 0C F4 D5 D4 9F D4 A4 59 7E 35 CF 32

6.2.2 AES Vector

Where 128-bit AES is used for content descrambling:

CW	68 E1 DA 5B 24 AD 86 1F 70 F9 C2 43 3C B5 9E 07
Scrambled Transport Stream Packet ($E_{CW}(Content)$)	47 00 71 90 34 8B 0C FE DF 26 16 CA BB 82 C2 8C 7F 70 6B 49 28 2D 08 23 89 45 F1 F2 98 7C A5 5C C2 96 C3 12 48 52 6A 8E 1E 1A 65 37 62 19 29 C8 F1 42 7F BE AE 67 94 B6 47 B6 C2 F5 77 52 48 03 1A E6 DB C2 7B 30 DB ED 90 A9 24 4D 26 92 11 80

	3F 9A DC 93 0B 7C 69 61 63 E6 41 1D DD B5 E1 01 BD A7 F4 46 D3 C2 D7 C7 FA 1A 9C B8 39 23 F5 57 E3 7B FC 4C 38 E8 DF 32 8F 70 F9 EF 3C 73 8F 01 23 7A 00 2F 7F 0D 88 C5 D9 F6 05 1C D0 D0 2D 4D FF 49 86 00 C2 0A D1 2D 85 C4 F6 47 C8 A9 97 0B 9E 49 AE 23 01 D1 44 41 6B FA 63 A6 FC 7D F9 13 16 23 28 9F DF 6E 87 0A 3B 11 2B BE
Clear Transport Stream Packet (*Content*)	47 00 71 10 23 BE 84 E1 6C D6 AE 52 90 49 F1 F1 BB E9 EB B3 A6 DB 3C 87 0C 3E 99 24 5E 0D 1C 06 B7 47 DE B3 12 4D C8 43 BB 8B A6 1F 03 5A 7D 09 38 25 1F 5D D4 CB FC 96 F5 45 3B 13 0D 89 0A 1C DB AE 32 20 9A 50 EE 40 78 36 FD 12 49 32 F6 9E 7D 49 DC AD 4F 14 F2 44 40 66 D0 6B C4 30 B7 32 3B A1 22 F6 22 91 9D E1 8B 1F DA B0 CA 99 02 B9 72 9D 49 2C 80 7E C5 99 D5 E9 80 B2 EA C9 CC 53 BF 67 D6 BF 14 D6 7E 2D DC 8E 66 83 EF 57 49 61 FF 69 8F 61 CD D1 1E 9D 9C 16 72 72 E6 1D F0 84 4F 4A 77 02 D7 E8 39 2C 53 CB C9 12 1E 33 74 9E 0C F4 D5 D4 9F D4 A4 59 7E 35 CF 32

Appendix A Test Vectors Python Script

```
from Crypto.Cipher import AES, DES3
import binascii

#=============================================
#CONSTANTS
#=============================================
FIXED_SCK = binascii.unhexlify('77656c636f6d65746f6d797061727479')
MASK_KEY = binascii.unhexlify('f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff')
EK2 = binascii.unhexlify('202122232425262728292a2b2c2d2e2f')
EK1 = binascii.unhexlify('101112131415161718191a1b1c1d1e1f')
NONCE = binascii.unhexlify('a0a1a2a3a4a5a6a7a8a9aaabacadaeaf')
VENDOR_ID = '\x2a\x42'
CW8 = binascii.unhexlify('BCFBB26913BABE8B')
CW8_2 = binascii.unhexlify('68E1DA5B24AD861F')
CW16 = binascii.unhexlify('68E1DA5B24AD861F70F9C2433CB59E07')
MODULE_ID = '\xa5'


#=============================================
#Utility functions
#=============================================

#pretty prints a string of bytes with a given separating character
def h(x,s=" "):
    return s.join(["%02X" % ord(c) for c in x])

#performs xor on two strings of the same length
def XOR(a,b):
    if len(a) != len(b):
        print 'XOR received strings of unequal length'

    result = ''.join([chr(ord(a[i]) ^ ord(b[i])) for i in range(len(a))])
    return result
#the following are needed for padding a CW out to 16 bytes, or no padding at all.
def pad8(cw):
```

```
    return cw + '\x00'*8
def padNone(cw):
    return cw
#============================================
#Crypto Functions
#============================================

def AESEncrypt(key,data):
    return AES.new(key,1).encrypt(data)

def AESDecrypt(key,data):
    return AES.new(key,1).decrypt(data)

def DES3Encrypt(key,data):
    return DES3.new(key,1).encrypt(data)

def DES3Decrypt(key,data):
    return DES3.new(key,1).decrypt(data)

#============================================
#These functions pad the vendor ID or module ID depending on the algorithm
#============================================
#for AES, just a straght zero-pad
def padVendorIDAES(vid):
    return '\x00' * 14 + vid

#for TDES, the vendor ID must be placed in each half, and we want to somehow make each half different
def padVendorIDDES3(vid):
    return '\x01' + '\x00' * 5 + vid + '\x02' + '\x00' *5 + vid

def padModuleIDAES(mid):
    return '\x00' * 15 + mid
#for TDES, the vendor ID must be placed in each half, and we want to somehow make each half different
def padModuleIDDES3(mid):
    return '\x01' + '\x00' * 6 + mid + '\x02' + '\x00' *6 + mid

#============================================
#The following functions implement the actual operations
#============================================
```

```
#There are two types of key derivations: with and without module ID. Within those groups
#the flow is the the same, it's just the algos and the VID/ModuleID padding that change.
#this function performs the basic operations
def BasicKeyDerivation(algo, SCK, vendorID):

    SCKv = algo(SCK,vendorID)
    Seedv = algo(MASK_KEY,vendorID)
    K3 = XOR(Seedv, algo(SCKv,Seedv))
    return {'SCKv':SCKv, 'Seedv':Seedv, 'K3':K3}

def BasicKeyDerivationWithModuleID(algo, SCK, vendorID,moduleID):
    SCKv = algo(SCK,vendorID)
    Seedv = algo(MASK_KEY,vendorID)
    Modv = XOR(Seedv, algo(SCKv,Seedv))
    K3 = algo(Modv, moduleID)
    return {'SCKv':SCKv, 'Seedv':Seedv, 'Modv':Modv, 'K3':K3}

#performs the CW KLAD path for the ETSI TS 103 162 standard
def BasicKLAD(algoD, algoE, K3, CW):
    K2 = algoD(K3,EK2)
    K1 = algoD(K2,EK1)
    ECW = algoE(K1,CW)
    return {'K3':K3, 'K2':K2,'K1':K1,'ECW':ECW,'CW':CW}

#performs the Challenge-response path for the ETSI TS 103 162 standard
def BasicCR(algoD, K3):
    K2 = algoD(K3,EK2)
    A = algoD(K2,K2)
    dnonce = algoD(A,NONCE)
    return {'K3':K3, 'K2':K2, 'A': A, 'dnonce':dnonce}

#=============================================
#Functions for pretty printing the vectors
#=============================================

def prettyPrintKDKeys(kdKeys,derivation):
    print 'Key Derivation Name: %s' % derivation['name']
    print 'SCK                : %s' % h(FIXED_SCK)
    print 'Mask Key           : %s' % h(MASK_KEY)
    print 'VID                : %s' % h(VENDOR_ID)
```

```
    print 'Padded VID         : %s' % h(derivation['vidPad'](VENDOR_ID))
    print 'SCKv               : %s' % h(kdKeys['SCKv'])
    print 'Seedv              : %s' % h(kdKeys['Seedv'])
    #check to see if there's a Modv key
    if 'Modv' in kdKeys:
        print 'MID                : %s' % h(MODULE_ID)
        print 'Padded MID         : %s' % h(derivation['midPad'](MODULE_ID))
        print 'Modv               : %s' % h(kdKeys['Modv'])
    print 'K3                 : %s' % h(kdKeys['K3'])

def prettyPrintKLADKeys(kladKeys,klad):
    print 'KLAD Algo         : %s' % klad['name']
    print 'K3                : %s' % h(kladKeys['K3'])
    print 'EK2               : %s' % h(EK2)
    print 'K2                : %s' % h(kladKeys['K2'])
    print 'EK1               : %s' % h(EK1)
    print 'K1                : %s' % h(kladKeys['K1'])
    print 'ECW               : %s' % h(kladKeys['ECW'])
    print 'CW                : %s' % h(kladKeys['CW'])

def prettyPrintCRKeys(crKeys, klad):
    print 'C/R Algo          : %s' % klad['name']
    print 'K3                : %s' % h(crKeys['K3'])
    print 'EK2               : %s' % h(EK2)
    print 'K2                : %s' % h(crKeys['K2'])
    print 'A                 : %s' % h(crKeys['A'])
    print 'nonce             : %s' % h(NONCE)
    print 'dnonce            : %s' % h(crKeys['dnonce'])

#=============================================
#These items describe the various key derivation/klad combinations we'll perform
#=============================================

AESKLAD = {'algoD': AESDecrypt, 'algoE': AESEncrypt, 'cw8Padder': pad8, 'name':'AES'}
DES3KLAD = {'algoD': DES3Decrypt, 'algoE': DES3Encrypt, 'cw8Padder': padNone, 'name': '3DES'}
keyDerivations = (
                {'algo':DES3Decrypt, 'vidPad': padVendorIDDES3, 'name': 'Profile 1: Triple DES
(decrypt) Profile'},
               {'algo':DES3Decrypt, 'vidPad': padVendorIDDES3, 'midPad':padModuleIDDES3, 'name':
'Profile 1a: Triple DES (decrypt) Profile with Module Key Derivation'},
```

```
               {'algo':AESEncrypt, 'vidPad': padVendorIDAES, 'name': 'Profile 2: AES (encrypt)
Profile'},
               {'algo':AESEncrypt, 'vidPad': padVendorIDAES, 'midPad':padModuleIDAES, 'name': 'Profile
2a: AES (encrypt) with Module Key Derivation'},
               {'algo':AESDecrypt, 'vidPad': padVendorIDAES, 'midPad':padModuleIDAES, 'name': 'Profile
2b: AES (decrypt) with Module Key Derivation'},

               )

#the main loop will iterate over the defined key derivation blocks
#for each key derivation, we use the resulting root key for all possible key ladder operations
#I.e., AES/TDES on both the CW and the C/R path.

for keyDerivation in keyDerivations:
    #we have slightly different handling for a key derivation depending on whether or not it supports
Module ID
    if 'midPad' in keyDerivation:
        kdKeys = BasicKeyDerivationWithModuleID(keyDerivation['algo'], FIXED_SCK,
keyDerivation['vidPad'](VENDOR_ID), keyDerivation['midPad'](MODULE_ID))
    else:
        kdKeys = BasicKeyDerivation(keyDerivation['algo'], FIXED_SCK,
keyDerivation['vidPad'](VENDOR_ID))
    print ("=================================================================================")
    print ("=================================================================================")
    prettyPrintKDKeys(kdKeys,keyDerivation)
    #iterate over KLAD algos
    for klad in (AESKLAD, DES3KLAD):
        for cw in (CW16, CW8, CW8_2):
            #if the cw is less than 16 bytes, we need to add extra padding for AES
            if len(cw) < 16:
                cw = klad['cw8Padder'](cw)
            #calculate the klad keys
            kladKeys = BasicKLAD(klad['algoD'], klad['algoE'], kdKeys['K3'], cw)
            print ("---------------------------------------------------------------------------------")
            prettyPrintKLADKeys(kladKeys,klad)
        #calculate Challenge/response keys.
        crKeys = BasicCR(klad['algoD'], kdKeys['K3'])
        print ("---------------------------------------------------------------------------------")
        prettyPrintCRKeys(crKeys,klad)
```