

3GPP2 C.S0084-005-0

Version 2.0

Date: August 2007



**3RD GENERATION
PARTNERSHIP
PROJECT 2
"3GPP2"**

Security Functions for Ultra Mobile Broadband (UMB) Air Interface Specification

COPYRIGHT

3GPP2 and its Organizational Partners claim copyright in this document and individual Organizational Partners may copyright and issue documents or standards publications in individual Organizational Partner's name based on this document. Requests for reproduction of this document should be directed to the 3GPP2 Secretariat at <mailto:secretariat@3gpp2.org>. Requests to reproduce individual Organizational Partner's documents should be directed to that Organizational Partner. See <http://www.3gpp2.org/> for more information.

No text.

CONTENTS

| | | |
|----|------------------------------------------------------------------------------------|-----|
| 1 | FOREWORD | ix |
| 2 | NOTES | xi |
| 3 | REFERENCES | xi |
| 4 | 1 Introduction | 1-1 |
| 5 | 2 AES CIPHERING PROTOCOL | 2-1 |
| 6 | 2.1 Primitives and Public Data | 2-1 |
| 7 | 2.1.1 Commands | 2-1 |
| 8 | 2.1.2 Return Indications | 2-1 |
| 9 | 2.1.3 Procedure Calls | 2-1 |
| 10 | 2.1.4 Local Common Data | 2-2 |
| 11 | 2.1.5 Public Data | 2-2 |
| 12 | 2.2 Protocol Data Unit | 2-2 |
| 13 | 2.3 Procedures and Messages for the InConfiguration Instance of the Protocol | 2-2 |
| 14 | 2.3.1 Protocol Initialization for the InConfiguration Protocol Instance | 2-2 |
| 15 | 2.3.2 Procedures | 2-2 |
| 16 | 2.3.3 Message Formats | 2-3 |
| 17 | 2.4 Procedures and Messages for the InUse Instance of the Protocol | 2-3 |
| 18 | 2.4.1 Procedures | 2-3 |
| 19 | 2.4.1.1 Protocol Initialization for the InUse Protocol Instance | 2-3 |
| 20 | 2.4.1.2 Hard Commit Procedures | 2-3 |
| 21 | 2.4.1.3 Soft Commit Procedures | 2-3 |
| 22 | 2.4.1.4 Constructing the Ciphering Key | 2-3 |
| 23 | 2.4.1.5 Constructing the Cryptosync | 2-4 |
| 24 | 2.4.1.6 Encrypt Procedures | 2-5 |
| 25 | 2.4.1.7 Decryption Procedures | 2-7 |
| 26 | 2.4.2 Message Formats | 2-9 |
| 27 | 2.4.3 Interface to Other Protocols | 2-9 |
| 28 | 2.4.3.1 Commands | 2-9 |
| 29 | 2.4.3.2 Indications | 2-9 |
| 30 | 2.5 Configuration Attributes | 2-9 |
| 31 | 2.5.1 Simple Attributes | 2-9 |

CONTENTS

| | | |
|----|-----------------------------------------------------------------------------------|------|
| 1 | 2.5.2 Complex Attributes | 2-10 |
| 2 | 2.5.2.1 FTCReducedStrengthCipherringKey Attribute | 2-10 |
| 3 | 2.5.2.2 RTCReducedStrengthCipherringKey Attribute | 2-10 |
| 4 | 2.6 Non-Attribute Data | 2-11 |
| 5 | 2.7 Protocol Numeric Constants | 2-11 |
| 6 | 2.8 Session State Information | 2-11 |
| 7 | 3 Basic Message Integrity Protocol | 3-1 |
| 8 | 3.1 3.1 Overview | 3-1 |
| 9 | 3.2 Primitives and Public Data | 3-1 |
| 10 | 3.2.1 Commands | 3-1 |
| 11 | 3.2.2 Return Indications | 3-1 |
| 12 | 3.2.3 Procedure Calls | 3-1 |
| 13 | 3.2.4 Local Common Data | 3-2 |
| 14 | 3.2.5 Public Data | 3-2 |
| 15 | 3.3 Protocol Data Unit..... | 3-2 |
| 16 | 3.4 Procedures and Messages for the InConfiguration Instance of the Protocol..... | 3-3 |
| 17 | 3.4.1 Protocol Initialization for the InConfiguration Protocol Instance..... | 3-3 |
| 18 | 3.4.2 Procedures | 3-3 |
| 19 | 3.4.3 Message Formats | 3-3 |
| 20 | 3.5 Procedures and Messages for the InUse Instance of the Protocol | 3-3 |
| 21 | 3.5.1 Procedures | 3-3 |
| 22 | 3.5.1.1 Protocol Initialization for the InUse Protocol Instance | 3-3 |
| 23 | 3.5.1.2 Hard Commit Procedures | 3-3 |
| 24 | 3.5.1.3 Soft Commit Procedures | 3-3 |
| 25 | 3.5.1.4 Constructing the Message Integrity Key | 3-3 |
| 26 | 3.5.1.5 Constructing the Cryptosync | 3-4 |
| 27 | 3.5.1.6 Authentication Header..... | 3-5 |
| 28 | 3.5.1.7 AUTHENTICATE_ADD_TAG procedures | 3-6 |
| 29 | 3.5.1.8 AUTHENTICATE_CHECK_TAG procedures | 3-7 |
| 30 | 3.5.1.9 CREATE_ID_TAG procedures..... | 3-9 |
| 31 | 3.5.2 Message Formats | 3-9 |
| 32 | 3.6 Interface to Other Protocols..... | 3-9 |

CONTENTS

| | | |
|----|------------------------------------------------------------------------------------|------|
| 1 | 3.6.1 Commands | 3-9 |
| 2 | 3.6.2 Indications | 3-9 |
| 3 | 3.7 Configuration Attributes | 3-9 |
| 4 | 3.8 Non-Attribute Data | 3-10 |
| 5 | 3.9 Protocol Numeric Constants | 3-10 |
| 6 | 3.10 Session State Information | 3-10 |
| 7 | 4 Basic Key Exchange Protocol | 4-1 |
| 8 | 4.1 Overview | 4-1 |
| 9 | 4.2 Primitives and Public Data | 4-1 |
| 10 | 4.2.1 Commands | 4-1 |
| 11 | 4.2.2 Return Indications | 4-1 |
| 12 | 4.2.3 Local Common Data | 4-1 |
| 13 | 4.2.4 Public Data | 4-1 |
| 14 | 4.2.5 Interface to Other Protocols | 4-2 |
| 15 | 4.2.5.1 Commands | 4-2 |
| 16 | 4.2.5.2 Indications | 4-2 |
| 17 | 4.3 Protocol Data Unit | 4-2 |
| 18 | 4.4 Procedures and Messages for the InConfiguration Instance of the Protocol | 4-2 |
| 19 | 4.4.1 Protocol Initialization for the InConfiguration Protocol Instance | 4-2 |
| 20 | 4.4.2 Procedures | 4-2 |
| 21 | 4.4.3 Message Formats | 4-2 |
| 22 | 4.5 Procedures and Messages for the InUse Instance of the Protocol | 4-2 |
| 23 | 4.5.1 Procedures | 4-2 |
| 24 | 4.5.1.1 Protocol Initialization for the InUse Protocol Instance | 4-3 |
| 25 | 4.5.1.2 Hard Commit Procedures | 4-3 |
| 26 | 4.5.1.3 Soft Commit Procedures | 4-3 |
| 27 | 4.5.1.4 Access Terminal Requirements | 4-3 |
| 28 | 4.5.1.4.1 Initiating the key exchange | 4-3 |
| 29 | 4.5.1.4.2 Processing a KeyResponse message | 4-4 |
| 30 | 4.5.1.4.3 Processing a KeyReject message | 4-5 |
| 31 | 4.5.1.5 Access Network Requirements | 4-5 |
| 32 | 4.5.1.5.1 Initiating the key exchange | 4-5 |

CONTENTS

1 4.5.1.5.2 Processing a KeyRequest message 4-5

2 4.5.1.5.3 Processing a KeyComplete message 4-5

3 4.5.1.6 MICKey Derivation..... 4-5

4 4.5.1.7 Message Integrity Key and Ciphering Key Generation 4-6

5 4.5.1.7.1 Temporary Security Key Derivation..... 4-6

6 4.5.1.7.2 Message Integrity Key and Ciphering Keys Generation from TSKey 4-6

7 4.5.1.8 EHMACH-SHA256(key, message, MAC_length)..... 4-6

8 4.5.2 Message Formats 4-7

9 4.5.2.1 KeyRequest..... 4-7

10 4.5.2.2 KeyResponse 4-7

11 4.5.2.3 KeyComplete 4-9

12 4.5.2.4 KeyReject..... 4-10

13 4.5.2.5 InitiateKeyRequest..... 4-11

14 4.5.3 Interface to Other Protocols 4-11

15 4.5.3.1 Commands 4-11

16 4.5.3.2 Indications 4-11

17 4.6 Configuration Attributes 4-11

18 4.7 Non-Attribute Data 4-11

19 4.8 Protocol Numeric Constants 4-12

20 4.9 Session State Information 4-12

21 4.9.1 PMK Parameter 4-12

22

FIGURES

1 Figure 3-1. AUTHENTICATE_ADD_TAG procedure call payloads3-6
2 Figure 3-2. AUTHENTICATE_CHECK_TAG procedure call payloads.....3-8
3

FIGURES

- 1 No text.

TABLES

1 Table 2-1. Subfield of the Cryptosync2-5
 2 Table 2-2. Configurable Values.....2-9
 3 Table 3-1. Subfield of the Cryptosync3-4
 4 Table 3-2. Authentication Headers3-5
 5 Table 3-3. AuthKeyIndex encoding3-5
 6 Table 3-4. Protocol Numeric Constants3-10
 7 Table 4-1. KeyRequest Message.....4-7
 8 Table 4-2. Definition of Result field4-10
 9 Table 4-3. Protocol Numeric Constants4-12
 10 Table 4-4. The Format of the Parameter Record for the PMK Parameter.....4-12
 11

TABLES

- 1 No text.

FOREWORD

1 **(This foreword is not part of this Standard)**

2 This standard was prepared by Technical Specification Group C of the Third Generation
3 Partnership Project 2 (3GPP2). This Standard is the Security Functions part of the Ultra
4 Mobile Broadband™ (UMB™)¹ air interface. Other parts of this Standard are:

- 5 • Overview for Ultra Mobile Broadband (UMB) Air Interface Specification
- 6 • Physical Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- 7 • MAC Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- 8 • Radio Link Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- 9 • Application Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- 10 • Connection Control Plane for Ultra Mobile Broadband (UMB) Air Interface Specification
- 11 • Session Control Plane for Ultra Mobile Broadband (UMB) Air Interface Specification
- 12 • Route Control Plane for Ultra Mobile Broadband (UMB) Air Interface Specification
- 13 • Broadcast-Multicast Upper Layers for Ultra Mobile Broadband (UMB) Air Interface
14 Specification

15 Other Standards may be required to implement this system and are listed in the References
16 section of each part.

17 This standard provides a specification for land mobile wireless systems based upon cellular
18 principles. This Standard is one part of the IMT-2000 CDMA Multi-Carrier, IMT-2000
19 CDMA MC, also known as cdma2000®².

¹ Ultra Mobile Broadband™ and (UMB™) are trade and service marks owned by the CDMA Development Group (CDG).

² cdma2000® is the trademark for the technical nomenclature for certain specifications and standards of the Organizational Partners (OPs) of 3GPP2. Geographically (and as of the date of publication), cdma2000® is a registered trademark of the Telecommunications Industry Association (TIA-USA) in the United States.

FOREWORD

- 1 No text.

REFERENCES

1 The following documents contain provisions, which, through reference in this text,
2 constitute provisions of this document. References are either specific (identified by date of
3 publication, edition number, version number, etc.) or non-specific. For a specific reference,
4 subsequent revisions do not apply. For a non-specific reference, the latest version applies.
5 In the case of a reference to a 3GPP2 document, a non-specific reference implicitly refers to
6 the latest version of that document in the same Release as the present document.
7

8 [1] C.S0084-000-0, Overview for Ultra Mobile Broadband (UMB) Air Interface
9 Specification.

10 [2] C.S0084-001-0, Physical Layer for Ultra Mobile Broadband (UMB) Air Interface
11 Specification.

12 [3] C.S0084-002-0, MAC Layer for Ultra Mobile Broadband (UMB) Air Interface
13 Specification.

14 [4] C.S0084-003-0, Radio Link Layer for Ultra Mobile Broadband (UMB) Air Interface
15 Specification.

16 [5] C.S0084-004-0, Application Layer for Ultra Mobile Broadband (UMB) Air Interface
17 Specification.

18 [6] Reserved.

19 [7] C.S0084-006-0, Connection Control Plane for Ultra Mobile Broadband (UMB) Air
20 Interface Specification.

21 [8] C.S0084-007-0, Session Control Plane for Ultra Mobile Broadband (UMB) Air
22 Interface Specification.

23 [9] C.S0084-008-0, Route Control Plane for Ultra Mobile Broadband (UMB) Air
24 Interface Specification.

25 [10] C.S0084-009-0, Broadcast-Multicast Upper Layer for Ultra Mobile Broadband
26 (UMB) Air Interface Specification.

27 [11] C.R1001, Administration of Parameter Value Assignments for cdma2000 Spread
28 Spectrum Standards. (Informative)

29 [12] S.S0055, Enhanced Cryptographic Algorithms.

30 [13] S.S0078, Common Security Algorithms.

31 [14] NIST, Special Publication 800-38B Draft, "Recommendation for Block Cipher
32 Modes of Operation: The CMAC Method for Authentication", March 9, 2005."

33 [15] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC
34 4493, June 2006."

35 [16] [CMAC-NIST-SP800-38B] NIST, Special Publication 800-38B, "Recommendation for
36 Block Cipher Modes of Operation: The CMAC Mode for Authentication", May 2005.

37 [17] IETF RFC 4493, The AES-CMAC Algorithm. (Informative)

REFERENCES

- 1 No text.

1 **1 INTRODUCTION**

2 Security Function consists of following protocols:

- 3 • AES Ciphering Protocol
- 4 • Basic Message Integrity Protocol
- 5 • Basic Key Exchange Protocol

- 1 No text.

1 **2 AES CIPHERING PROTOCOL**

2 The AES Ciphering Protocol uses the AES (a.k.a. Rijndael) procedures defined in [12] in
3 order to encrypt and decrypt the Radio Link Protocol packets.

4 **2.1 Primitives and Public Data**

5 2.1.1 Commands

6 This protocol does not define any commands.

7 2.1.2 Return Indications

8 This protocol does not return any indications.

9 2.1.3 Procedure Calls

10 • ENCRYPT

- 11 – Inputs: *Direction*, *DataUnit*, *StreamID*, *RouteCounter*, *SARResetCounter*,
- 12 *SARSequenceNumber*, *SARSequenceRolloverCounter*, *PayloadSize*, *Payload*
- 13 – Outputs: *Payload* (Encrypted), *CipheringKeyIndex*
- 14 – Possible values of each input and output are as follows:
 - 15 + *Direction* – ‘0’ (ForwardLink) or ‘1’ (ReverseLink)
 - 16 + *DataUnit* – ‘00’ (Octets), ‘01’ (RLP Packet Payload)
 - 17 + *StreamID* – 16-bit hexadecimal number
 - 18 + *RouteCounter* – 15-bit hexadecimal number
 - 19 + *SARResetCounter* – 8-bit hexadecimal number
 - 20 + *SARSequenceNumber* – Hexadecimal number of n bits, where n is less than or
 - 21 equal to 48.
 - 22 + *SARSequenceRolloverCounter* – Hexadecimal number of (48 – length of
 - 23 *SARSequenceNumber*) bits.
 - 24 + *PayloadSize* – Payload size in octets
 - 25 + *Payload* – Payload to be encrypted
 - 26 + *Payload* (Encrypted) – Encrypted payload of same size as input Payload and is
 - 27 available at same memory space as the input Payload.
 - 28 + *CipheringKeyIndex* – ‘00’ (not encrypted); ‘01’, ‘10’, ‘11’ (encrypted with key
 - 29 corresponding to index ‘01’, ‘10’ or ‘11’ respectively)

30 • DECRYPT

- 31 – Inputs: *Direction*, *DataUnit*, *StreamID*, *RouteCounter*, *SARResetCounter*,
- 32 *SARSequenceNumber*, *SARSequenceRolloverCounter*, *PayloadSize*, *Payload*,
- 33 *CipheringKeyIndex*

- 1 – Outputs: *Payload* (Decrypted)
- 2 – Possible values of each input and output are as follows:
- 3 + *Direction* – ‘0’ (ForwardLink) or ‘1’ (ReverseLink)
- 4 + *DataUnit* – ‘00’ (Octets), ‘01’ (RLP Packet Payload)
- 5 + *StreamID* – 16-bit hexadecimal number
- 6 + *RouteCounter* – 15-bit hexadecimal number
- 7 + *SARResetCounter* – 8-bit hexadecimal number
- 8 + *SARSequenceNumber* – Hexadecimal number of n bits, where n is less than or
- 9 equal to 48.
- 10 + *SARSequenceRolloverCounter* – Hexadecimal number of (48 – length of
- 11 *SARSequenceNumber*) bits.
- 12 + *PayloadSize* – Payload size in octets
- 13 + *Payload* – Payload to be decrypted
- 14 + *CipheringKeyIndex* – ‘00’ (not encrypted); ‘01’, ‘10’, ‘11’ (encrypted with key
- 15 corresponding to index ‘01’, ‘10’ or ‘11’ respectively)
- 16 + *Payload* (Decrypted) – Decrypted payload of same size as input Payload and is
- 17 available at same memory space as the input Payload.

18 2.1.4 Local Common Data

19 This protocol does not define any Local Common Data.

20 2.1.5 Public Data

21 This protocol shall make the following data public:

- 22 • Subtype for this protocol
- 23 • All data defined as Static Attribute, Static Non-Attribute Data, and Local Common Data

24 **2.2 Protocol Data Unit**

25 This protocol does not transmit or receive data. This protocol provides encryption and

26 decryption services to Radio Link Protocol.

27 **2.3 Procedures and Messages for the InConfiguration Instance of the Protocol**

28 2.3.1 Protocol Initialization for the InConfiguration Protocol Instance

29 Upon creation, the InConfiguration instance of this protocol in the access terminal and the

30 access network shall perform the procedures specified in [1].

31 2.3.2 Procedures

32 This protocol uses the services of the Session Control Protocol to perform negotiation of

33 attribute values.

1 2.3.3 Message Formats

2 This protocol does not define any messages.

3 **2.4 Procedures and Messages for the InUse Instance of the Protocol**

4 2.4.1 Procedures

5 2.4.1.1 Protocol Initialization for the InUse Protocol Instance

6 Upon creation, the InUse instance of this protocol in the access terminal and access
7 network shall perform the procedures specified in [1].

8 2.4.1.2 Hard Commit Procedures

9 The access terminal and the access network shall perform the procedures specified in
10 [1]when directed by the InUse instance of the Session Control Protocol to execute the Hard
11 Commit procedures.

12 2.4.1.3 Soft Commit Procedures

13 The access terminal and the access network shall perform the procedures specified in [1]
14 when directed by the InUse instance of the Session Control Protocol to execute the Soft
15 Commit procedures.

16 2.4.1.4 Constructing the Ciphering Key

17 The AES Ciphering Protocol shall construct the Ciphering keys as follows:

- 18 • If the value of the FTCCiphering attribute is equal to 0x01, then the protocol shall
19 construct the Ciphering key for the Forward Traffic Channel, FTCCipheringKey, as
20 follows:
 - 21 – If the FACCipheringKey[KeyIndex] public data of the Key Exchange Protocol is set to
22 NULL, the protocol shall set FTCCipheringKey to NULL.
 - 23 – Otherwise, the protocol shall perform the following:
 - 24 + If the length of FACCipheringKey[KeyIndex] is equal to 128, then
25 FTCCipheringKey shall be set to FACCipheringKey[KeyIndex].
 - 26 + Otherwise, if the length of FACCipheringKey[KeyIndex] is greater than 128, then
27 FTCCipheringKey shall be the 128 most significant bits of
28 FACCipheringKey[KeyIndex].
 - 29 + Otherwise, if the length of FACCipheringKey[KeyIndex] is less than 128, then
30 FTCCipheringKey shall be the concatenation of zeros at the end (LSB) of
31 FACCipheringKey[KeyIndex], such that the length of the result is 128.
 - 32 – The protocol shall perform the following:
 - 33 + Call the KeyStrengthRedAlg procedure specified in [13] with its inputs set as
34 follows:
 - 35 ▪ Set the *KeyLength* to 16.

- 1 ▪ Set the *OriginalKey* to the value of the FTCCipheringKey.
- 2 ▪ Set the *SaltLength* to the value of the FTCSaltLength parameter.
- 3 ▪ Set the *Salt* to the value of the FTCSalt parameter.
- 4 ▪ Set the *KeyEntropy* to the value of the FTCKeyEntropy parameter.
- 5 + When the KeyStrengthRedAlg procedure returns, set the FTCCipheringKey to
- 6 *RedStrengthKey* which is the output of the KeyStrengthRedAlg procedure.
- 7 • If the value of the RTCCiphering attribute is equal to 0x01, then the protocol shall
- 8 construct the Ciphering key for the Reverse Traffic Channel, RTCCipheringKey, as
- 9 follows:
- 10 – If the RACCipheringKey[KeyIndex] public data of the Key Exchange Protocol is set to
- 11 NULL, the protocol shall set RTCCipheringKey to NULL.
- 12 – Otherwise, the protocol shall perform the following:
- 13 + If the length of RACCipheringKey[KeyIndex] is equal to 128, then
- 14 RTCCipheringKey shall be set to RACCipheringKey[KeyIndex].
- 15 + Otherwise, if the length of RACCipheringKey[KeyIndex] is greater than 128, then
- 16 RTCCipheringKey shall be the 128 most significant bits of
- 17 RACCipheringKey[KeyIndex].
- 18 + Otherwise, if the length of RACCipheringKey[KeyIndex] is less than 128, then
- 19 RTCCipheringKey shall be the concatenation of zeros at the end (LSB) of
- 20 RACCipheringKey[KeyIndex], such that the length of the result is 128.
- 21 – The protocol shall perform the following:
- 22 + Call the KeyStrengthRedAlg procedure specified in [13] with its inputs set as
- 23 follows:
- 24 ▪ Set the *KeyLength* to 16.
- 25 ▪ Set the *OriginalKey* to the value of the RTCCipheringKey.
- 26 ▪ Set the *SaltLength* to the value of the RTCSaltLength parameter.
- 27 ▪ Set the *Salt* to the value of the RTCSalt parameter.
- 28 ▪ Set the *KeyEntropy* to the value of the RTCKeyEntropy parameter.
- 29 + When the KeyStrengthRedAlg procedure returns, set the RTCCipheringKey to
- 30 *RedStrengthKey* which is the output of the KeyStrengthRedAlg procedure.

31 2.4.1.5 Constructing the Cryptosync

32 The protocol shall construct the Cryptosync as shown in Table 2-1.

1

Table 2-1. Subfield of the Cryptosync

| Subfield | Length (bits) |
|--------------------------|---------------|
| FunctionCode | 2 |
| Reserved | 6 |
| Direction | 1 |
| RouteCounter | 15 |
| StreamID | 16 |
| SARResetCounter | 8 |
| VirtualSARSequenceNumber | 48 |

- 2 FunctionCode This field shall be set to '00' to indicate Ciphering function.
- 3 Reserved All the bits in this field shall be set to '0'.
- 4 Direction If the payload being encrypted or decrypted is for Forward
5 Link then this field shall be set to '0'. Otherwise, this field
6 shall be set to '1'. Direction is received as *Direction* input to
7 ENCRYPT and DECRYPT procedure calls.
- 8 RouteCounter This field shall be set to the RouteCounter corresponding to
9 the payload being encrypted or decrypted. RouteCounter is
10 received as *RouteCounter* input to ENCRYPT and DECRYPT
11 procedure calls.
- 12 StreamID This field shall be set to the StreamID corresponding to the
13 payload being encrypted or decrypted. StreamID is received as
14 *StreamID* input to ENCRYPT and DECRYPT procedure calls.
- 15 SARResetCounter This field shall be set to SARResetCounter corresponding to
16 the payload being encrypted or decrypted. SARResetCounter
17 is received as *SARResetCounter* input to ENCRYPT and
18 DECRYPT procedure calls.
- 19 VirtualSARSequenceNumber This field shall be set to VirtualSARSequenceNumber
20 corresponding to the payload being encrypted or decrypted.
21 VirtualSARSequenceNumber value is set as specified in
22 sections 2.4.1.6 and 2.4.1.7.

23 2.4.1.6 Encrypt Procedures

24 The protocol shall provide encryption services through ENCRYPT procedure call.

25 If any of the following conditions is true:

- 26 • The Ciphering attribute for the channel under consideration (e.g., FTCCiphering) is
27 equal to 0x00.

- 1 • The KeyIndex public data of the Key Exchange Protocol is set to NULL.
- 2 • The CipherringKey for the channel under consideration (e.g., FTCCipherringKey),
3 constructed as specified in 2.4.1.4 where KeyIndex is set to KeyIndex public data of the
4 Key Exchange Protocol, is NULL.
- 5 Then, the ENCRYPT procedure shall set the outputs of the ENCRYPT procedure as follows:
- 6 • Set *Payload* to the input *Payload*.
- 7 • Set *CipherringKeyIndex* to '00'.
- 8 Otherwise, the ENCRYPT procedure shall perform the following:
- 9 • If *DataUnit* is set to '01' then the ENCRYPT procedure shall perform the following:
- 10 – The ENCRYPT procedure shall call the ESP_AES procedure specified in [12] with its
11 inputs set as follows:
- 12 + Set the *key* to the CipherringKey for the channel under consideration (e.g.,
13 FTCCipherringKey) constructed as specified in 2.4.1.4 where KeyIndex is set to
14 KeyIndex public data of the Key Exchange Protocol.
- 15 + Set *fresh* to the value of the Cryptosync constructed as specified in 2.4.1.5,
16 where VirtualSARSequenceNumber is set to (*SARSequenceRolloverCounter* |
17 *SARSequenceNumber*).
- 18 + Set the *freshsize* to the length of the Cryptosync in octets.
- 19 + Set the *buf* to the address of the beginning of the memory space that contains
20 the *Payload*.
- 21 + Set the *bit_offset* to zero.
- 22 + Set the *bit_count* to 8 times *PayloadSize*.
- 23 – After the ESP_AES procedure is returned, the ENCRYPT procedure shall set the
24 outputs of the ENCRYPT procedure as follows:
- 25 + Set *Payload* to the output of the ESP_AES procedure which starts at the memory
26 space specified by *buf* and is of the same size as the input *Payload*.
- 27 + Set *CipherringKeyIndex* to KeyIndex public data of the Key Exchange Protocol.
- 28 • If *DataUnit* is set to '00' then the ENCRYPT procedure shall perform the following:
- 29 – Set VirtualSARSequenceNumberInUse to (*SARSequenceRolloverCounter* |
30 *SARSequenceNumber*)
- 31 – Set *n* to 1.
- 32 – The ENCRYPT procedure shall perform following steps *PayloadSize* times³:

³ Even though steps here are performed on each octet of payload, in implementation the steps can be performed on data blocks of 16 octets of payload except first and last data blocks which could have fewer octets.

- 1 + Call the ESP_AES procedure specified in [12] with its inputs set as follows:
- 2 ▪ Set the *key* to the CipherringKey for the channel under consideration (e.g.,
- 3 FTCCipherringKey) constructed as specified in 2.4.1.4 where KeyIndex is set
- 4 to KeyIndex public data of the Key Exchange Protocol.
- 5 ▪ Set *fresh* to the value of the Cryptosync constructed as specified in 2.4.1.5,
- 6 where VirtualSARSequenceNumber is set to $16 \times \lfloor$
- 7 VirtualSARSequenceNumberInUse / 16 \rfloor .
- 8 ▪ Set the *freshsize* to the length of the Cryptosync in octets.
- 9 ▪ Create a *temp_buf* and initialize it to (VirtualSARSequenceNumberInUse mod
- 10 16) octets of 0x00 followed by the nth octet of *Payload*. Set the *buf* to the
- 11 address of the beginning of the memory space that contains the *temp_buf*.
- 12 ▪ Set the *bit_offset* to zero.
- 13 ▪ Set the *bit_count* to $\lceil (\text{VirtualSARSequenceNumberInUse mod } 16) + 1 \rceil \times 8$.
- 14 + After the ESP_AES procedure is returned, the ENCRYPT procedure shall skip
- 15 first (VirtualSARSequenceNumberInUse mod 16) octets of the output of the
- 16 ESP_AES procedure which starts at the memory space specified by *temp_buf*,
- 17 and overwrite nth octet of the *Payload* with the value of the next octet of the
- 18 ESP_AES procedure output.
- 19 + Increment value of n
- 20 + Increment value of VirtualSARSequenceNumberInUse modulo 2^{48} .
- 21 – The ENCRYPT procedure shall set the outputs of the ENCRYPT procedure as
- 22 follows:
- 23 + Set *CipherringKeyIndex* to KeyIndex public data of the Key Exchange Protocol.

24 2.4.1.7 Decryption Procedures

25 The protocol shall provide decryption services through DECRYPT procedure call.

26 If the Cipherring attribute for the channel under consideration (e.g., FTCCipherring) is equal

27 to 0x01 and *CipherringKeyIndex* is not set to '00', the DECRYPT procedure shall perform the

28 following:

- 29 • If *DataUnit* is set to '01' then the DECRYPT procedure shall perform the following:
- 30 – The DECRYPT procedure shall call the ESP_AES procedure specified in [12] with its
- 31 inputs set as follows:
- 32 + Set the *key* to the CipherringKey for the channel under consideration (e.g.,
- 33 FTCCipherringKey) constructed as specified in 2.4.1.4 where KeyIndex is set to
- 34 *CipherringKeyIndex*.
- 35 + Set *fresh* to the value of the Cryptosync constructed as specified in 2.4.1.5,
- 36 where VirtualSARSequenceNumber is set to $(\text{SARSequenceRolloverCounter} \mid$
- 37 *SARSequenceNumber* $)$.
- 38 + Set the *freshsize* to the length of the Cryptosync in octets.

- 1 + Set the *buf* to the address of the beginning of the memory space that contains
2 the *Payload*.
- 3 + Set the *bit_offset* to zero.
- 4 + Set the *bit_count* to 8 times *PayloadSize*.
- 5 – After the ESP_AES procedure is returned, the DECRYPT procedure shall set the
6 outputs of the DECRYPT procedure as follows:
- 7 + Set *Payload* to the output of the ESP_AES procedure which starts at the memory
8 space specified by *buf* and is of the same size as the input *Payload*.
- 9 • If *DataUnit* is set to '00' then the DECRYPT procedure shall perform the following:
- 10 – Set *VirtualSARSequenceNumberInUse* to (*SARSequenceRolloverCounter* |
11 *SARSequenceNumber*).
- 12 – Set *n* to 1.
- 13 – The DECRYPT procedure shall perform following steps *PayloadSize* times⁴:
- 14 + Call the ESP_AES procedure specified in [12] with its inputs set as follows:
- 15 ▪ Set the *key* to the *CipheringKey* for the channel under consideration (e.g.,
16 *FTCCipheringKey*) constructed as specified in 2.4.1.4 where *KeyIndex* is set
17 to *CipheringKeyIndex*.
- 18 ▪ Set *fresh* to the value of the *Cryptosync* constructed as specified in 2.4.1.5,
19 where *VirtualSARSequenceNumber* is set to $16 \times \lfloor$
20 $\text{VirtualSARSequenceNumberInUse} / 16 \rfloor$.
- 21 ▪ Set the *freshsize* to the length of the *Cryptosync* in octets.
- 22 ▪ Create a *temp_buf* and initialize it to (*VirtualSARSequenceNumberInUse* mod
23 16) octets of 0x00 followed by the *n*th octet of *Payload*. Set the *buf* to the
24 address of the beginning of the memory space that contains the *temp_buf*.
- 25 ▪ Set the *bit_offset* to zero.
- 26 ▪ Set the *bit_count* to $\lfloor (\text{VirtualSARSequenceNumberInUse} \bmod 16) + 1 \rfloor \times 8$.
- 27 + After the ESP_AES procedure is returned, the DECRYPT procedure shall skip
28 first (*VirtualSARSequenceNumberInUse* mod 16) octets of the output of the
29 ESP_AES procedure which starts at the memory space specified by *temp_buf*,
30 and overwrite *n*th octet of the *Payload* with the value of the next octet of the
31 ESP_AES procedure output.
- 32 + Increment value of *n*.
- 33 + Increment value of *VirtualSARSequenceNumberInUse* modulo 2^{48} .

⁴ Even though steps here are performed on each octet of payload, in implementation the steps can be performed on data blocks of 16 octets of payload except first and last data blocks which could have fewer octets.

1 Otherwise, the DECRYPT procedure shall set the outputs of the DECRYPT procedure as
2 follows:

- 3 • Set *Payload* to the input *Payload*

4 2.4.2 Message Formats

5 No messages are defined for the InUse instance of this protocol.

6 2.4.3 Interface to Other Protocols

7 2.4.3.1 Commands

8 This protocol does not issue any commands.

9 2.4.3.2 Indications

10 This protocol does not register to receive any indications.

11 2.5 Configuration Attributes

12 The following attributes and default values are defined (see [1] for attribute record
13 definition).

14 2.5.1 Simple Attributes

15 The configurable simple attributes for this protocol are listed in the Table 2-2.

16 The default value for each attribute is typed in ***bold italics***.

17

Table 2-2. Configurable Values

| Attribute ID | Attribute | Commit/Scope | Values | Meaning |
|--------------|--------------|--------------|--------------------|-------------------------------------------------------------------------------------------------------------------|
| 0x0000 | FTCCiphering | Soft/Dynamic | 0x00 | RLP packets destined for the FTC shall not be encrypted by the sender and shall not be decrypted by the receiver. |
| | | | <i>0x01</i> | RLP packets destined for the FTC shall be encrypted by the sender and shall be decrypted by the receiver. |
| | | | 0x02-0xff | Reserved |
| 0x0001 | RTCCiphering | Soft/Dynamic | 0x00 | RLP packets destined for the RTC shall not be encrypted by the sender and shall not be decrypted by the receiver. |
| | | | <i>0x01</i> | RLP packets destined for the RTC shall be encrypted by the sender and shall be decrypted by the receiver. |
| | | | 0x02-0xff | Reserved |

1 2.5.2 Complex Attributes

2 The following complex attributes are defined for reduction of the Ciphering key strength for
3 each channel.

4 2.5.2.1 FTCSaltLengthCipheringKey Attribute

5 The sender shall set AttributeID field to 0x8000.
6

| Field | Length (bits) | Default Value |
|---------------|-------------------|---------------|
| FTCSaltLength | 8 | 0 |
| FTCSalt | FTCSaltLength × 8 | N/A |
| FTCKeyEntropy | 8 | 16 |

7 FTCSaltLength The sender shall set this field to the length of the FTCSalt
8 field in octets.

9 FTCSalt The sender shall set this field to the value of the *Salt* input
10 parameter that is to be used in the KeyStrengthRedAlg
11 procedure specified in [13] for the FTC Ciphering key.

12 FTCKeyEntropy The sender shall set this field to the value of the *KeyEntropy*
13 input parameter that is to be used in the KeyStrengthRedAlg
14 procedure specified in [13] for the FTC Ciphering key. The
15 valid values for this field are 0 through 16, inclusive.
16

| | | | |
|---------------|------|--------------|---------|
| Commit | Hard | Scope | Dynamic |
|---------------|------|--------------|---------|

17 2.5.2.2 RTCReducedStrengthCipheringKey Attribute

18 The sender shall set AttributeID field to 0x8001.
19

| Field | Length (bits) | Default Value |
|---------------|-------------------|---------------|
| RTCSaltLength | 8 | 0 |
| RTCSalt | RTCSaltLength × 8 | N/A |
| RTCKeyEntropy | 8 | 16 |

20 RTCSaltLength The sender shall set this field to the length of the RTCSalt
21 field in octets.

22 RTCSalt The sender shall set this field to the value of the *Salt* input
23 parameter that is to be used in the KeyStrengthRedAlg
24 procedure specified in [13] for the RTC Ciphering key.

25 RTCKeyEntropy The sender shall set this field to the value of the *KeyEntropy*
26 input parameter that is to be used in the KeyStrengthRedAlg

1 procedure specified in [13] for the RTC Ciphering key. The
 2 valid values for this field are 0 through 16, inclusive.
 3

| | | | |
|---------------|------|--------------|---------|
| Commit | Hard | Scope | Dynamic |
|---------------|------|--------------|---------|

4 **2.6 Non-Attribute Data**

5 This protocol does not define any non-attribute data.

6 **2.7 Protocol Numeric Constants**

7

| Constant | Meaning | Value |
|----------------------|---------------------------------|-------|
| N _{CPT} ype | Type field for this protocol | [1] |
| N _{CP} AES | Subtype field for this protocol | 0x00 |

8 **2.8 Session State Information**

9 The Session State Information record (see [1]) consists of parameter records.

10 All configuration attributes and Non-attribute data are Session State Information records.

11 This protocol does not define additional parameter records.

- 1 No text.

1 **3 BASIC MESSAGE INTEGRITY PROTOCOL**

2 **3.1 3.1 Overview**

3 The Basic Message Integrity Protocol provides a method for integrity protection of signaling
4 messages by applying the AES CMAC function (see [14] and [15]).

5 **3.2 Primitives and Public Data**

6 3.2.1 Commands

7 This protocol does not define any commands.

8 3.2.2 Return Indications

9 This protocol does not define any indications.

10 3.2.3 Procedure Calls

- 11 • **AUTHENTICATE_ADD_TAG**
 - 12 – Inputs: *Direction*, *StreamID*, *RouteCounter*, *SARResetCounter*,
13 *SARSequenceNumber*, *SARSequenceRolloverCounter*, *InputPayloadSize*,
14 *InputPayload*
 - 15 – Outputs: *OutputPayloadSize*, *OutputPayload*
 - 16 – Possible values of each input and output are as follows:
 - 17 + *Direction* – ‘0’ (*ForwardLink*) or ‘1’ (*ReverseLink*)
 - 18 + *StreamID* – 16-bit hexadecimal number
 - 19 + *RouteCounter* – 15-bit hexadecimal number
 - 20 + *SARResetCounter* – 8-bit hexadecimal number
 - 21 + *SARSequenceNumber* – Hexadecimal number of *n* bits, where *n* is less than or
22 equal to 48
 - 23 + *SARSequenceRolloverCounter* – Hexadecimal number of (48 – length of
24 *SARSequenceNumber*) bits
 - 25 + *InputPayloadSize* – Input payload size in octets
 - 26 + *InputPayload* – Input payload
 - 27 + *OutputPayloadSize* – Output payload size in octets
 - 28 + *OutputPayload* – Output payload
- 29 • **AUTHENTICATE_CHECK_TAG**
 - 30 – Inputs: *Direction*, *StreamID*, *RouteCounter*, *SARResetCounter*, *SARSequenceNumber*,
31 *SARSequenceRolloverCounter*, *InputPayloadSize*, *InputPayload*
 - 32 – Outputs: *TagMatched*, *OutputPayloadSize*, *OutputPayload*

- 1 – Possible values of each input and output are as follows:
- 2 + *Direction* – ‘0’ (ForwardLink) or ‘1’ (ReverseLink)
- 3 + *StreamID* – 16-bit hexadecimal number
- 4 + *RouteCounter* – 15-bit hexadecimal number
- 5 + *SARResetCounter* – 8-bit hexadecimal number
- 6 + *SARSequenceNumber* – Hexadecimal number of n bits, where n is less than or
- 7 equal to 48.
- 8 + *SARSequenceRolloverCounter* – Hexadecimal number of (48 – length of
- 9 *SARSequenceNumber*) bits.
- 10 + *InputPayloadSize* – Input payload size in octets
- 11 + *InputPayload* – Input payload
- 12 + *TagMatched* – ‘1’ (TRUE), ‘0’ (FALSE)
- 13 + *OutputPayloadSize* – Output payload size in octets
- 14 + *OutputPayload* – Output payload
- 15 • CREATE_ID_TAG
- 16 – Inputs: SequenceNumber
- 17 – Outputs: IDTagSize, IDTag
- 18 – Possible values of each input and output are as follows:
- 19 + *SequenceNumber* – 32-bit hexadecimal number
- 20 + *IDTagSize* – ID Tag size in octets
- 21 + *IDTag* – ID Tag

22 3.2.4 Local Common Data

23 This protocol does not define any Local Common Data.

24 3.2.5 Public Data

25 This protocol shall make the following data public:

- 26 • Subtype for this protocol
- 27 • All data defined as Static Attribute, Static Non-Attribute Data, and Local Common Data

28 3.3 Protocol Data Unit

29 This protocol does not transmit or receive data. This protocol provides integrity protection
30 services to Radio Link Protocol.

1 **3.4 Procedures and Messages for the InConfiguration Instance of the Protocol**

2 3.4.1 Protocol Initialization for the InConfiguration Protocol Instance

3 Upon creation, the InConfiguration instance of this protocol in the access terminal and the
4 access network shall perform the procedures specified in [1].

5 3.4.2 Procedures

6 This protocol uses the services of the Session Control Protocol to perform negotiation of
7 attribute values.

8 3.4.3 Message Formats

9 This protocol does not define any messages.

10 **3.5 Procedures and Messages for the InUse Instance of the Protocol**

11 3.5.1 Procedures

12 3.5.1.1 Protocol Initialization for the InUse Protocol Instance

13 Upon creation, the InUse instance of this protocol in the access terminal and access
14 network shall perform the procedures specified in [1].

15 3.5.1.2 Hard Commit Procedures

16 The access terminal and the access network shall perform the procedures specified in [1]
17 when directed by the InUse instance of the Session Control Protocol to execute the Hard
18 Commit procedures.

19 3.5.1.3 Soft Commit Procedures

20 The access terminal and the access network shall perform the procedures specified in [1]
21 when directed by the InUse instance of the Session Control Protocol to execute the Soft
22 Commit procedures.

23 3.5.1.4 Constructing the Message Integrity Key

24 Basic Message Integrity Protocol shall construct the Message Integrity keys as follows:

- 25 • The protocol shall construct the Message Integrity key for the Forward Traffic Channel,
26 FTCKMIKey as follows:
 - 27 – If the FACMKIKey[KeyIndex] public data of the Key Exchange Protocol is set to NULL,
28 the protocol shall set FTCKMIKey to NULL.
 - 29 – Otherwise, the protocol shall perform the following:
 - 30 + If the length of FACMKIKey[KeyIndex] is equal to 128 bits, then FTCKMIKey shall
31 be set to FACMKIKey[KeyIndex].
 - 32 + Otherwise, if the length of FACMKIKey[KeyIndex] is greater than 128 bits, then
33 FTCKMIKey shall be the 128 most significant bits of FACMKIKey[KeyIndex].

- 1 + Otherwise, if the length of FACMIKey[KeyIndex] is less than 128 bits, then
 2 FTCTMIKey shall be the concatenation of zeros at the end (LSB) of
 3 FACMIKey[KeyIndex], such that the length of the result is 128 bits.
- 4 • The protocol shall construct the Message Integrity key for the Reverse Traffic Channel,
 5 RTCTMIKey as follows:
- 6 – If the RACMIKey[KeyIndex] public data of the Key Exchange Protocol is set to NULL,
 7 the protocol shall set RTCTMIKey to NULL.
- 8 – Otherwise, the protocol shall perform the following:
- 9 + If the length of RACMIKey[KeyIndex] is equal to 128 bits, then RTCTMIKey shall
 10 be set to RACMIKey[KeyIndex].
- 11 + Otherwise, if the length of RACMIKey[KeyIndex] is greater than 128 bits, then
 12 RTCTMIKey shall be the 128 most significant bits of RACMIKey[KeyIndex].
- 13 + Otherwise, if the length of RACMIKey[KeyIndex] is less than 128 bits, then
 14 RTCTMIKey shall be the concatenation of zeros at the end (LSB) of
 15 RACMIKey[KeyIndex], such that the length of the result is 128 bits.

16 3.5.1.5 Constructing the Cryptosync

17 The protocol shall construct the Cryptosync as shown in Table 3-1.

18 **Table 3-1. Subfield of the Cryptosync**

| Subfield | Length (bits) |
|--------------------------|---------------|
| Direction | 1 |
| RouteCounter | 15 |
| StreamID | 16 |
| SARResetCounter | 8 |
| VirtualSARSequenceNumber | 48 |

- 19 Reserved All the bits in this field shall be set to '0'.
- 20 Direction If the payload is for Forward Link then this field shall be set
 21 to '0'. Otherwise, this field shall be set to '1'. Direction is
 22 received as Direction input to AUTHENTICATE_ADD_TAG and
 23 AUTHENTICATE_CHECK_TAG procedure calls.
- 24 RouteCounter This field shall be set to the RouteCounter corresponding to
 25 the payload. RouteCounter is received as RouteCounter input
 26 to AUTHENTICATE_ADD_TAG and
 27 AUTHENTICATE_CHECK_TAG procedure calls.
- 28 StreamID This field shall be set to the StreamID corresponding to the
 29 payload. StreamID is received as StreamID input to

1 AUTHENTICATE_ADD_TAG and
 2 AUTHENTICATE_CHECK_TAG procedure calls.

3 SARResetCounter This field shall be set to SARResetCounter corresponding to
 4 the payload. SARResetCounter is received as
 5 SARResetCounter input to AUTHENTICATE_ADD_TAG and
 6 AUTHENTICATE_CHECK_TAG procedure calls.

7 VirtualSARSequenceNumber This field shall be set to VirtualSARSequenceNumber
 8 corresponding to the payload. VirtualSARSequenceNumber
 9 value is set as specified in sections 3.5.1.7 and 3.5.1.8.

10 3.5.1.6 Authentication Header

11 The Authentication Header, which precedes the payload, has the following format, as
 12 shown in Table 3-2.

13 **Table 3-2. Authentication Headers**

| Field | Length (bits) |
|--------------|---------------|
| AuthKeyIndex | 2 |
| Reserved | 6 |
| AuthTag | 0 or 64 |

14 AuthKeyIndex The protocol shall set this field to indicate index of the key
 15 used for authentication of the payload as specified in Table
 16 3-3.

17 **Table 3-3. AuthKeyIndex encoding**

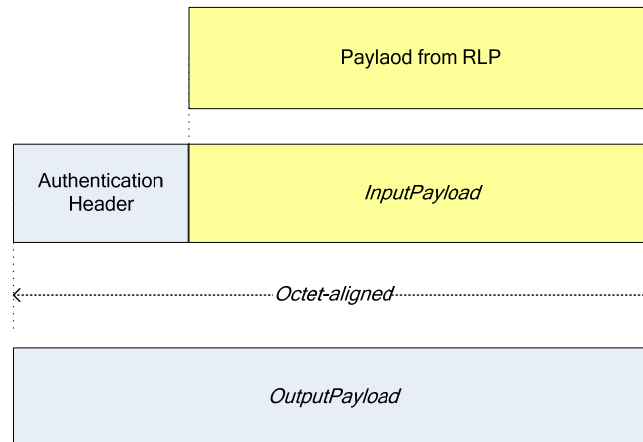
| AuthKeyIndex | Meaning |
|--------------|-------------------------------------------------------|
| '00' | AuthTag not included |
| '01' | AuthTag computed with key corresponding to index '01' |
| '10' | AuthTag computed with key corresponding to index '10' |
| '11' | AuthTag computed with key corresponding to index '11' |

18 Reserved The protocol shall set all the bits in this field to '0'.

19 AuthTag The protocol shall omit this field if the AuthKeyIndex field is
 20 set to '00'; otherwise, the protocol shall include this field and
 21 set it to authentication tag.

1 3.5.1.7 AUTHENTICATE_ADD_TAG procedures

2 The protocol shall provide service of adding authentication tag through
 3 AUTHENTICATE_ADD_TAG procedure call. Figure 3-1 illustrates the relationship between
 4 an InputPayload and an OutputPayload of AUTHENTICATE_ADD_TAG procedure call.



5

6

Figure 3-1. AUTHENTICATE_ADD_TAG procedure call payloads

7 If any of the following conditions is true:

- 8 • The KeyIndex public data of the Key Exchange Protocol is set to NULL.
- 9 • The MIKey for the channel under consideration (e.g., FTCTMIKey), constructed as
 10 specified in 3.5.1.4 where KeyIndex is set to KeyIndex public data of the Key Exchange
 11 Protocol, is NULL.
- 12 • Payload contains a message for which authentication tag is not required (See [1] and
 13 each protocol text) and if protocol decides not to include the authentication tag.

14 then, the AUTHENTICATE_ADD_TAG procedure shall set the outputs of the
 15 AUTHENTICATE_ADD_TAG procedure as follows:

- 16 • Construct Authentication Header with the AuthKeyIndex field set to '00'.
- 17 • Set *OutputPayload* to (Authentication Header | *InputPayload*).
- 18 • Set *OutputPayloadSize* to (*InputPayloadSize* + size of the Authentication Header in
 19 octets).

20 Otherwise, the AUTHENTICATE_ADD_TAG procedure shall perform the following:

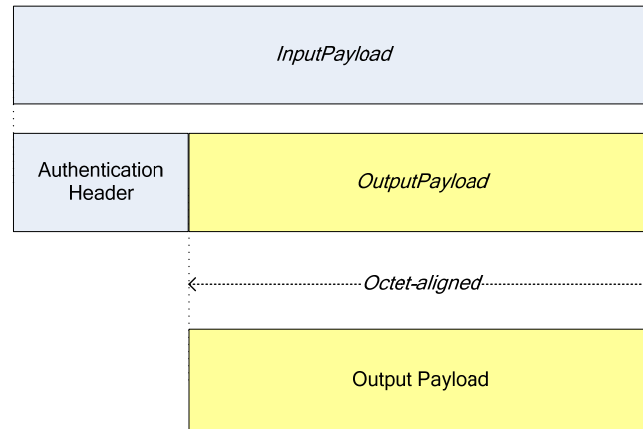
- 21 • The AUTHENTICATE_ADD_TAG procedure shall call the CMAC(K, M, Mlen, Tlen)
 22 procedure specified in Section 6.2 of [16] with its inputs set as follows:
- 23 – Set the *K* to the MIKey for the channel under consideration (e.g., FTCTMIKey)
 24 constructed as specified in 3.5.1.4 where KeyIndex is set to KeyIndex public data of
 25 the Key Exchange Protocol.

- 1 – Set the M to $BOHeader | B0 | InputPayload$, where $BOHeader$ and $B0$ are set as
2 follows:
- 3 + Set $BOHeader$ to ‘10011100’⁵.
- 4 + Set $B0$ to $(fresh | L)$, where $fresh$ is set to the value of the Cryptosync
5 constructed as specified in 3.5.1.5 with $VirtualSARSequenceNumber$ set to
6 $(SARSequenceRolloverCounter | SARSequenceNumber)$, and L is set to size of the
7 $InputPayload$ in octets expressed as 32-bit hexadecimal number.
- 8 – Set the $Mlen$ to 8 times size of M in octets.
- 9 – Set the $Tlen$ to 64.
- 10 • After the CMAC procedure is returned, the AUTHENTICATE_ADD_TAG procedure shall
11 set the outputs of the AUTHENTICATE_ADD_TAG procedure as follows:
- 12 – Construct Authentication Header with fields set as follows:
- 13 + Set the AuthTag field to the output of the CMAC procedure.
- 14 + Set the AuthKeyIndex field to KeyIndex public data of the Key Exchange
15 Protocol.
- 16 – Set $OutputPayload$ to $(Authentication\ Header | InputPayload)$
- 17 – Set $OutputPayloadSize$ to $(InputPayloadSize + \text{size of the Authentication Header in}$
18 octets)

19 3.5.1.8 AUTHENTICATE_CHECK_TAG procedures

20 The protocol shall provide service of checking authentication tag through
21 AUTHENTICATE_CHECK_TAG procedure call. Figure 3-2 illustrates the relationship
22 between an $InputPayload$ and an $OutputPayload$ of AUTHENTICATE_CHECK_TAG
23 procedure call.

⁵ Bits 5,4,3 of $BOHeader$ are encoded as $(M-2)/2$; where MSB of $BOHeader$ is bit 7, and M is $Tlen/8$ and can take the values 4, 6, 8, 10, 12, 14 and 16. Presently $Tlen$ is always 64, and hence bits 5, 4, 3 or $BOHeader$ are always set to 011.



1

2

Figure 3-2. AUTHENTICATE_CHECK_TAG procedure call payloads

3 The AUTHENTICATE_CHECK_TAG procedure shall perform the following:

- 4 • The *InputPayload* consists of (Authentication Header | *OutputPayload*).
- 5 • The AUTHENTICATE_CHECK_TAG procedure shall remove Authentication Header from
- 6 *InputPayload* to produce *OutputPayload*.
- 7 • If the AuthKeyIndex field of the Authentication Header is set to '00', then
- 8 AUTHENTICATE_CHECK_TAG procedure shall set the outputs of the
- 9 AUTHENTICATE_CHECK_TAG procedure as follows:
- 10 – Set *TagMatched* to TRUE.
- 11 – Set *OutputPayloadSize* to size of *OutputPayload* in octets.
- 12 • Otherwise, the AUTHENTICATE_CHECK_TAG procedure shall call the CMAC(K, M,
- 13 Mlen, Tlen) procedure specified in Section 6.2 of [16] with its inputs set as follows:
- 14 – Set the *K* to the MIKey for the channel under consideration (e.g., FTCMIKey)
- 15 constructed as specified in 3.5.1.4 where KeyIndex is set to AuthKeyIndex field of
- 16 the Authentication Header.
- 17 – Set the *M* to B0Header | B0 | *OutputPayload*, where B0Header and B0 are set as
- 18 follows:
- 19 + Set B0Header to '10011100'⁶.
- 20 + Set B0 to (fresh | L), where fresh is set to the value of the Cryptosync
- 21 constructed as specified in 3.5.1.5 with VirtualSARSequenceNumber set to
- 22 (SARSequenceRolloverCounter | SARSequenceNumber), and L is set to size of the
- 23 *OutputPayload* in octets expressed as 32-bit hexadecimal number.

⁶ Bits 5,4,3 of B0Header are encoded as $(M-2)/2$; where MSB of B0Header is bit 7, and M is Tlen/8 and can take the values 4, 6, 8, 10, 12, 14 and 16. Presently Tlen is always 64, and hence bits 5, 4, 3 or B0Header are always set to 011.

- 1 – Set the *Mlen* to 8 times size of *M* in octets.
- 2 – Set the *Tlen* to 64.
- 3 • After the CMAC procedure is returned, the AUTHENTICATE_CHECK_TAG procedure
- 4 shall set the outputs of the AUTHENTICATE_CHECK_TAG procedure as follows:
- 5 – If output of the CMAC procedure matches the AuthTag field of the Authentication
- 6 Header, then set *TagMatched* to TRUE. Otherwise, set *TagMatched* to FALSE.
- 7 – Set *OutputPayloadSize* to size of *OutputPayload* in octets.

8 3.5.1.9 CREATE_ID_TAG procedures

9 The protocol shall provide service of creating IDTag CREATE_ID_TAG procedure call.

10 The CREATE_ID_TAG procedure shall perform the following:

- 11 • If *KeyIndex* or *RACMIKey*[*KeyIndex*] public data of the Key Exchange Protocol is set to
- 12 NULL, then set the outputs of the CREATE_ID_TAG procedure as follows:
- 13 – Set *IDTagSize* to 0
- 14 – Set *IDTag* to NULL
- 15 • Otherwise, perform the following:
- 16 – Set *MIKeyID* to EHMAC-SHA256(key=*RACMIKey*[*KeyIndex*], message= “*MIKeyID*”,
- 17 MAC_length=4), where *RACMIKey*[*KeyIndex*] and *KeyIndex* are public data of the
- 18 Key Exchange Protocol and EHMAC-SHA256 function is specified in 4.5.1.8.
- 19 – Set *Tag* to EHMAC-SHA256(key=*RACMIKey*[*KeyIndex*], message= *SequenceNumber*,
- 20 MAC_length=8), where *RACMIKey*[*KeyIndex*] and *KeyIndex* are public data of the
- 21 Key Exchange Protocol, EHMAC-SHA256 function is specified in 4.5.1.8.
- 22 – Set the outputs of the CREATE_ID_TAG procedure as follows:
- 23 + Set *IDTagSize* to 12
- 24 + Set *IDTag* to (*MIKeyID* | *Tag*)

25 3.5.2 Message Formats

26 No messages are defined for the InUse instance of this protocol.

27 **3.6 Interface to Other Protocols**

28 3.6.1 Commands

29 This protocol does not issue any commands.

30 3.6.2 Indications

31 This protocol does not register to receive any indications.

32 **3.7 Configuration Attributes**

33 This protocol does not define any attributes.

1 **3.8 Non-Attribute Data**

2 This protocol does not define any non-attribute data.

3 **3.9 Protocol Numeric Constants**

4 **Table 3-4. Protocol Numeric Constants**

| Constant | Meaning | Value |
|---------------|---------------------------------|-------|
| $N_{MIPTYPE}$ | Type field for this protocol | [1] |
| N_{BMIP} | Subtype field for this protocol | 0x00 |

5 **3.10 Session State Information**

6 The Session State Information record (see [1]) consists of parameter records.

7 All configuration attributes and Non-attribute data are Session State Information records.

8 This protocol does not define additional parameter records.

1 **4 BASIC KEY EXCHANGE PROTOCOL**

2 **4.1 Overview**

3 The Basic Key Exchange Protocol provides a method for generating security keys at the
4 access terminal and the access network based on a Pairwise Master Key. The Pairwise
5 Master Key is established by higher layer protocols.

6 The Basic Key Exchange Protocol performs the following functions:

- 7 • Proves that both access terminal and access network have the same Pairwise Master
8 Key.
- 9 • Derives security keys from the Pairwise Master Key.
- 10 • Protects against a man-in-the-middle attack where a rogue entity causes the access
11 terminal and the access network to agree upon a weaker security protocol.

12 **4.2 Primitives and Public Data**

13 4.2.1 Commands

14 This protocol does not define any commands.

15 4.2.2 Return Indications

16 This protocol does not return any indications.

17 4.2.3 Local Common Data

18 This protocol defines the following Local Common Data (i.e., AT only):

- 19 • PMKList[]
20 A list of all Pairwise Master Keys.

21 4.2.4 Public Data

22 This protocol shall make the following data public:

- 23 • Subtype for this protocol
- 24 • LatestPMK (access terminal only)
25 Latest PMK established or used on this route.
- 26 • LatestPMKCreationTime (access terminal only)
27 Creation time of the LatestPMK
- 28 • KeyIndex
29 The key index in use. Valid values are '01', '10', and '11'.
- 30 • FACMIKey[i] and its length for values of i '01' through '11'
31 The Message Integrity key for use on Forward Assigned Channels (e.g., the Forward
32 Traffic Channel).

- 1 • RACMIKey[i] and its length for values of i '01' through '11'
- 2 The Message Integrity key for use on Reverse Assigned Channels (e.g., the Reverse
- 3 Traffic Channel).
- 4 • FACCipheringKey[i] and its length for values of i '01' through '11'
- 5 The Ciphering key for use on Forward Assigned Channels (e.g., the Forward Traffic
- 6 Channel).
- 7 • RACCipheringKey[i] and its length for values of i '01' through '11'
- 8 The Ciphering key for use on Reverse Assigned Channels (e.g., the Reverse Traffic
- 9 Channel).
- 10 • All data defined as Static Attribute, Static Non-Attribute Data, and Local Common Data

11 4.2.5 Interface to Other Protocols

12 4.2.5.1 Commands

13 This protocol does not define any commands.

14 4.2.5.2 Indications

15 This protocol does not register to receive any indications.

16 **4.3 Protocol Data Unit**

17 The transmission unit of this protocol is a message. This is a control protocol and,
18 therefore, it does not carry payload on behalf of other layers or protocols.

19 This protocol uses the Signaling Protocol to transmit and receive messages.

20 **4.4 Procedures and Messages for the InConfiguration Instance of the Protocol**

21 4.4.1 Protocol Initialization for the InConfiguration Protocol Instance

22 Upon creation, the InConfiguration instance of this protocol in the access terminal and the
23 access network shall perform the procedures specified in [1].

24 4.4.2 Procedures

25 This protocol uses the services of the Session Control Protocol to perform negotiation of
26 attribute values

27 4.4.3 Message Formats

28 This protocol does not define any messages.

29 **4.5 Procedures and Messages for the InUse Instance of the Protocol**

30 4.5.1 Procedures

31 The Basic Key Exchange Protocol derives security keys and exchanges security capabilities
32 as well as security protocols in use. The key exchange procedure uses the KeyRequest,
33 KeyResponse, KeyComplete, KeyReject, and InitiateKeyRequest messages.

1 4.5.1.1 Protocol Initialization for the InUse Protocol Instance

2 Upon creation, the InUse instance of this protocol in the access terminal and the access
3 network shall perform the following in the order specified:

- 4 • Perform the procedures specified in [1].
- 5 • Access terminal shall Set LatestPMK to NULL.
- 6 • Access terminal shall set LatestPMKCreationTime to NULL.
- 7 • Set KeyIndex to NULL.
- 8 • Set FACMIKey[i] to NULL, for values of i '01' through '11'.
- 9 • Set RACMIKey[i] to NULL, for values of i '01' through '11'.
- 10 • Set FACCipheringKey[i] to NULL, for values of i '01' through '11'.
- 11 • Set RACCipheringKey[i] to NULL, for values of i '01' through '11'.

12 4.5.1.2 Hard Commit Procedures

13 The access terminal and the access network shall perform the procedures specified in [1]
14 when directed by the InUse instance of the Session Control Protocol to execute the Hard
15 Commit procedures.

16 4.5.1.3 Soft Commit Procedures

17 The access terminal and the access network shall perform the procedures specified in [1]
18 when directed by the InUse instance of the Session Control Protocol to execute the Soft
19 Commit procedures.

20 4.5.1.4 Access Terminal Requirements

21 When a Pairwise Master Key is established by higher layer protocols, the access terminal
22 shall perform the following:

- 23 • Add the established PMK to the PMKList along with creation time of the key.
- 24 • Set LatestPMK to the established PMK
- 25 • Set LatestPMKCreationTime to creation time of the established PMK.

26 4.5.1.4.1 Initiating the key exchange

27 Access terminal may initiate the key exchange procedure by sending a KeyRequest
28 message. Access terminal should send a KeyRequest message when a new route open
29 procedure is initiated.

30 Upon receiving the InitiateKeyRequest message, the access terminal shall send a
31 KeyRequest message unless access terminal has already sent a KeyRequest message and is
32 awaiting the response.

1 4.5.1.4.2 Processing a KeyResponse message

2 Upon receiving the KeyResponse message with a TransactionID field that matches the
3 TransactionID field of the associated KeyRequest message, the access terminal shall
4 perform the following in the order in which the requirements are specified:

- 5 • The access terminal shall identify the PairwiseMasterKey from PMKList of any route
6 that satisfies PairwiseMasterKeyID = EHMAC-SHA256(key=PairwiseMasterKey,
7 message= "PairwiseMasterKeyID", MAC_length=8), where PairwiseMasterKeyID is a field
8 of the received KeyResponse message, "PairwiseMasterKeyID" is the ASCII encoded
9 value of the string, and the EHMAC-SHA256 function is specified in 4.5.1.8.
- 10 • If the access terminal cannot identify a valid PairwiseMasterKey that satisfies the above
11 condition, then the access terminal shall send a KeyComplete message with Result set
12 to 0x03, declare failure, and stop performing the rest of the key exchange procedure.
- 13 • The access terminal shall generate MICKey as specified in 4.5.1.6.
- 14 • The access terminal shall generate a MessageIntegrityCode as EHMAC-
15 SHA256(key=MICKey, message=Message, MAC_length=16), where Message is the
16 received KeyResponse message with the MessageIntegrityCode field set to zero, and the
17 EHMAC-SHA256 function is specified in 4.5.1.8.
- 18 • If the MessageIntegrityCode computed in the previous step does not match the
19 MessageIntegrityCode field of KeyResponse message, then the access terminal shall
20 declare failure, send a KeyComplete message with Result set to 0x01, and stop
21 performing the rest of the key exchange procedure.
- 22 • If the supported ProtocolSetIdentifiers sent by the access network in the KeyResponse
23 message do not match with the ProtocolSetIdentifiers supported by the access terminal
24 (i.e., either all the ProtocolSetIdentifiers supported by the access terminal are not
25 included in the KeyResponse message or KeyResponse message includes a
26 ProtocolSetIdentifiers that is not supported by the access terminal), then the access
27 terminal shall declare failure, send a KeyComplete message with Result set to 0x02,
28 and stop performing the rest of the key exchange procedure.
- 29 • If the first ProtocolSetIdentifier field sent by the access network in the KeyResponse
30 message do not match with the ProtocolSetIdentifier currently in use by the access
31 terminal, then the access terminal shall declare failure, send a KeyComplete message
32 with Result set to 0x04, and stop performing the rest of the key exchange procedure.
- 33 • The access terminal shall generate Message Integrity Keys and Ciphering Keys as
34 specified in 4.5.1.7 for KeyIndex value specified in the KeyResponse message. The
35 access terminal shall set KeyIndex public data to KeyIndex value specified in the
36 KeyResponse message.
- 37 • The access terminal shall send a KeyComplete message with Result set to 0x00.
- 38 • If LatestPMKCreationTime is set to NULL or if creation time of PMK identified by the
39 PairwiseMasterKeyID field of the received KeyResponse message is more recent than
40 time indicated by LatestPMKCreationTime, then access terminal shall perform the
41 following:

- 1 – Set LatestPMK to PMK identified by the PairwiseMasterKeyID field of the received
- 2 KeyResponse message.
- 3 – Set LatestPMKCreationTime to creation time of LatestPMK.

4 4.5.1.4.3 Processing a KeyReject message

5 Upon receiving the KeyReject message with a TransactionID field that matches the
6 TransactionID field of the associated KeyRequest message, the access terminal shall
7 terminate the key exchange procedure.

8 4.5.1.5 Access Network Requirements

9 4.5.1.5.1 Initiating the key exchange

10 Access network may initiate the key exchange procedure by sending an InitiateKeyRequest
11 message.

12 4.5.1.5.2 Processing a KeyRequest message

13 Upon receiving the KeyRequest message, the access network shall perform the following in
14 the order in which the requirements are specified:

- 15 • The access network shall send a KeyResponse message or a KeyReject message.
- 16 • If a KeyResponse message was sent, the access network shall generate MICKey as
17 specified in 4.5.1.6.

18 4.5.1.5.3 Processing a KeyComplete message

19 After receiving a KeyComplete message with a TransactionID field that matches the
20 TransactionID field of the associated KeyResponse message, the access network shall
21 perform the following:

- 22 • The access network shall generate a MessageIntegrityCode as EHMACHMAC-
23 SHA256(key=MICKey, message=Message, MAC_length=16), where Message is the
24 received KeyComplete message with the MessageIntegrityCode field set to zero, and the
25 EHMACHMAC-SHA256 function is specified in 4.5.1.8.
- 26 • If the MessageIntegrityCode computed in the previous step does not match the
27 MessageIntegrityCode field of KeyComplete message, then the access network shall
28 declare failure, and shall not use Message Integrity Keys and Ciphering Keys generated
29 in this key exchange procedure.
- 30 • If the Result field of the KeyComplete message is not 0x00, then the access network
31 shall declare failure and shall not use Message Integrity Keys and Ciphering Keys
32 generated in this key exchange procedure.

33 4.5.1.6 MICKey Derivation

34 The access terminal and the access network shall derive MICKey as follows:

- 35 • Set MICKey to zero and its length to 128.

- 1 • Set MICKey to EHMACHA256(key=PairwiseMasterKey, message=
2 “MICKey”|ATNonce|ANNonce, MAC_length=16), where “MICKey” is the ASCII encoded
3 value of the string, and the EHMACHA256 function is specified in 4.5.1.8.

4 4.5.1.7 Message Integrity Key and Ciphering Key Generation

5 The access terminal and the access network shall generate a TSKey as specified in
6 4.5.1.7.1. The access terminal and the access network shall generate Message Integrity and
7 Ciphering keys from TSKey as specified in 4.5.1.7.2.

8 4.5.1.7.1 Temporary Security Key Derivation

9 The access terminal and the access network shall derive TSKey as follows:

- 10 • Set TSKey to zero and its length to $N_{\text{BKEPTKeyLen}}$.
11 • Set j to an 8-bit number with value zero.
12 • while $j < N_{\text{BKEPTKeyLen}}/256$
13 – Set TSKey to $N_{\text{BKEPTKeyLen}}$ least significant bits of { TSKey | EHMACHA256-
14 SHA256(key=PairwiseMasterKey, message= “TSK”| ATNonce|ANNonce| j ,
15 MAC_length=32) }, where “TSK” is the ASCII encoded value of the string, j is
16 represented as an 8-bit field, and the EHMACHA256 function is specified in
17 4.5.1.8.
18 – Set j to $j+1$.

19 4.5.1.7.2 Message Integrity Key and Ciphering Keys Generation from TSKey

20 The keys used for Message Integrity and Ciphering are generated from the temporary
21 security key using the procedures specified in this section.

22 The access network and the access terminal shall compute and store Message Integrity
23 Keys and Ciphering Keys for KeyIndex i as follows:

- 24 • The access network and the access terminal shall set FACMIKey[i] to TSKey[127:0].
25 • The access network and the access terminal shall set RACMIKey[i] to TSKey[127:0].
26 • The access network and the access terminal shall set FACCIpheringKey[i] to
27 TSKey[255:128].
28 • The access network and the access terminal shall set RACCIpheringKey[i] to
29 TSKey[255:128].

30 4.5.1.8 EHMACHA256(key, message, MAC_length)

31 The EHMACHA256 procedure shall call ehmacsha256 procedure as specified in [13] with
32 following inputs:

- 33 • Set the *key_length* to the length of the key in bits,
34 • Set the *key* to the key,
35 • Set the *message* to the message, and

- 1 • Set the *message_length* to the length of the message in bits,
 - 2 • Set the *message_offset* to 0,
 - 3 • Set the *MAC_length* to 8 × *MAC_length*.
- 4 The output of the EHMAC-SHA256 function shall be set to the output of the ehmacsha256
5 procedure.

6 4.5.2 Message Formats

7 4.5.2.1 KeyRequest

8 The access terminal sends the KeyRequest message to initiate the session key exchange.

9 **Table 4-1. KeyRequest Message**

| Field | Length (bits) |
|---------------|---------------|
| MessageID | 8 |
| TransactionID | 8 |
| ATNonce | 128 |

- 10 MessageID The access terminal shall set this field to 0x00.
- 11 TransactionID The access terminal shall increment this value for each new
12 KeyRequest message sent.
- 13 ATNonce The access terminal shall set this field to a 128-bit random
14 number.
15

| | |
|-------------------|---------|
| Channels | RTC |
| Addressing | unicast |

| | |
|----------------|--------------------------------|
| RLP | Reliable |
| AuthTag | Required when key is available |

16 4.5.2.2 KeyResponse

17 The access network may send the KeyResponse message in response to the KeyRequest
18 message.
19

| Field | Length (bits) |
|---------------------------|---------------|
| MessageID | 8 |
| TransactionID | 8 |
| KeyIndex | 2 |
| Reserved | 6 |
| PairwiseMasterKeyID | 64 |
| ANNonce | 128 |
| NumProtocolSetIdentifiers | 8 |

NumProtocolSetIdentifiers occurrences of the following field:

| | |
|-----------------------|----|
| ProtocolSetIdentifier | 16 |
|-----------------------|----|

| | |
|----------------------|-----|
| MessageIntegrityCode | 128 |
|----------------------|-----|

- 1 MessageID The access network shall set this field to 0x01.
- 2 TransactionID The access network shall set this field to the value of the
3 TransactionID field of the KeyRequest message to which the
4 access network is responding.
- 5 KeyIndex The access network shall set this field to key index for which
6 this key exchange is being initiated. Valid values for this field
7 are '01' through '11'. The access network shall not set this
8 field to KeyIndex public data.
- 9 Reserved The access network shall set all the bits in this field to '0'. The
10 access terminal shall ignore this field.
- 11 PairwiseMasterKeyID The access network shall set this field to EHMAL-
12 SHA256(key=PairwiseMasterKey, message=
13 "PairwiseMasterKeyID", MAC_length=8), where
14 PairwiseMasterKey is a PairwiseMasterKey to be used,
15 "PairwiseMasterKeyID" is the ASCII encoded value of the
16 string, and the EHMAL-SHA256 function is specified in
17 4.5.1.8.
- 18 ANNonce The access network shall set this field to a 128-bit random
19 number.
- 20 NumProtocolSetIdentifiers The access network shall set this field to the number of
21 ProtocolSetIdentifiers supported by the access terminal.
22 ProtocolSetIdentifiers supported by the access terminal are
23 known to the access network through session information.

- 1 ProtocolSetIdentifier The access network shall set this field to the
 2 ProtocolSetIdentifier supported by the access terminal. The
 3 access network shall include the ProtocolSetIdentifier
 4 currently in use at the beginning. Protocol subtypes
 5 supported by the access terminal are known to the access
 6 network through session information.
- 7 MessageIntegrityCode The access network shall set this field to EHM
 8 AC-SHA256(key=MICKey, message=Message, MAC_length=16),
 9 where Message is set to all fields of this message with this
 10 field set to zero, and the EHM
 11 AC-SHA256 function is specified
 12 in 4.5.1.8.

| | |
|-------------------|---------|
| Channels | FTC |
| Addressing | unicast |

| | |
|----------------|--------------------------------|
| RLP | Reliable |
| AuthTag | Required when key is available |

13 4.5.2.3 KeyComplete

- 14 The access terminal sends the KeyComplete message in response to the KeyResponse
 15 message.
 16

| Field | Length (bits) |
|----------------------|---------------|
| MessageID | 8 |
| TransactionID | 8 |
| Result | 8 |
| MessageIntegrityCode | 0 or 128 |

- 17 MessageID The access terminal shall set this field to 0x02.
- 18 TransactionID The access terminal shall set this field to the value of the
 19 TransactionID field of the corresponding KeyRequest message.
- 20 Result The access terminal shall set this field according to Table 4-1.

1

Table 4-2. Definition of Result field

| Value | Meaning |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| 0x00 | Key exchange successful. |
| 0x01 | MessageIntegrityCode failed |
| 0x02 | MessageIntegrityCode successful, but ProtocolSetIdentifiers verification failed. |
| 0x03 | PairwiseMasterKey not found. |
| 0x04 | MessageIntegrityCode successful, ProtocolSetIdentifiers verification successful, but verification of ProtocolSetIdentifier in use failed. |
| All other values | Reserved |

2 MessageIntegrityCode If Result is 0x00, then the access terminal shall set this field
3 to EHMAL-SHA256(key=MICKey, message=*Message*,
4 MAC_length=16), where *Message* is set to all fields of this
5 message with this field set to zero, and the EHMAL-SHA256
6 function is specified in 4.5.1.8. Otherwise, the access terminal
7 shall omit this field.
8

| | |
|-------------------|---------|
| Channels | RTC |
| Addressing | unicast |

| | |
|----------------|--------------------------------|
| RLP | Reliable |
| AuthTag | Required when key is available |

9 4.5.2.4 KeyReject

10 The access network may send the KeyReject message in response to the KeyRequest
11 message.
12

| Field | Length (bits) |
|---------------|---------------|
| MessageID | 8 |
| TransactionID | 8 |

13 MessageID The access network shall set this field to 0x03.

- 1 TransactionID The access network shall set this field to the value of the
 2 TransactionID field of the KeyRequest message to which the
 3 access network is responding.
 4

| | | | |
|-------------------|---------|----------------|--------------------------------|
| Channels | FTC | RLP | Reliable |
| Addressing | unicast | AuthTag | Required when key is available |

5 4.5.2.5 InitiateKeyRequest

- 6 The access network may send the InitiateKeyRequest message to request the access
 7 terminal to send a KeyRequest message.
 8

| Field | Length (bits) |
|-----------|---------------|
| MessageID | 8 |

- 9 MessageID The access network shall set this field to 0x04.
 10

| | | | |
|-------------------|---------|----------------|--------------------------------|
| Channels | FTC | RLP | Reliable |
| Addressing | unicast | AuthTag | Required when key is available |

11 4.5.3 Interface to Other Protocols

12 4.5.3.1 Commands

- 13 This protocol does not issue any commands.

14 4.5.3.2 Indications

- 15 This protocol does not register to receive any indications.

16 **4.6 Configuration Attributes**

- 17 This protocol does not define any attributes.

18 **4.7 Non-Attribute Data**

- 19 This protocol does not define any non-attribute data.

1 4.8 Protocol Numeric Constants

2 **Table 4-3. Protocol Numeric Constants**

| Constant | Meaning | Value |
|---------------------------|---------------------------------------------------|-------|
| N_{KEPTType} | Type field for this protocol | [1] |
| N_{BKEP} | Subtype field for this protocol | 0x00 |
| $N_{\text{BKEPTSKeyLen}}$ | Length of Temporary Security Key in units of bits | 256 |

3 4.9 Session State Information

4 The Session State Information record (see [1]) consists of parameter records.

5 All configuration attributes and Non-attribute data are Session State Information records.
6 This protocol defines the following parameter record in addition to the configuration
7 attributes for this protocol.

8 4.9.1 PMK Parameter

9 The sender shall set DataID field to 0x0000.

10 **Table 4-4. The Format of the Parameter Record for the PMK Parameter**

| Field | Length (bits) |
|----------------|---------------|
| ValidPMKExists | 1 |
| Reserved | 7 |
| PMKCount | 8 |

PMKCount occurrences of the following five fields:

| | |
|-------------------|---------------|
| PMKLength | 8 |
| PMK | PMKLength × 8 |
| PMKGenerationTime | 24 |
| PMKMinLifeTime | 24 |
| PMKMaxLifeTime | 24 |

11 ValidPMKExists If a valid PMK exists, then this field shall be set to '1';
12 otherwise, this field shall be set to '0'.

13 Reserved All the bits in this field shall be set to '0'.

14 PMKCount This field shall be set to the number of occurrences of the
15 PMK field in this parameter record.

16 PMKLength This field shall be set to the length of the PMK field in units of
17 octets.

| | | |
|---|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 1 | PMK | This field shall be set to a PairwiseMasterKey. |
| 2 | PMKGenerationTime | This field shall be set to $X \bmod 2^{24}$; where X is the time, in units of seconds, at which the PairwiseMasterKey was generated. |
| 3 | | |
| 4 | | |
| 5 | PMKMinLifeTime | This field shall be set to minimum lifetime of the PairwiseMasterKey in units of seconds. |
| 6 | | |
| 7 | PMKMaxLifeTime | This field shall be set to maximum lifetime of the PairwiseMasterKey in units of seconds. |
| 8 | | |

- 1 No text.