

3GPP2 C.S0084-005-0

Version 1.0

Date: April, 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	xi
2	NOTES	xiii
3	REFERENCES	xiii
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 Protocol Initialization	2-2
14	2.3.1 Protocol Initialization for the InConfiguration Protocol Instance	2-2
15	2.3.2 Protocol Initialization for the InUse Protocol Instance	2-2
16	2.4 Procedures and Messages for the InConfiguration Instance of the Protocol	2-3
17	2.4.1 Procedures	2-3
18	2.4.2 Message Formats	2-3
19	2.5 Procedures and Messages for the InUse Instance of the Protocol	2-3
20	2.5.1 Procedures	2-3
21	2.5.1.1 Hard Commit Procedures	2-3
22	2.5.1.2 Soft Commit Procedures	2-3
23	2.5.1.3 Constructing the Ciphering Key	2-3
24	2.5.1.4 Constructing the Cryptosync	2-4
25	2.5.1.5 Encrypt Procedures	2-5
26	2.5.1.6 Decryption Procedures	2-7
27	2.5.2 Message Formats	2-9
28	2.5.3 Interface to Other Protocols	2-9
29	2.5.3.1 Commands	2-9
30	2.5.3.2 Indications	2-9
31	2.6 Configuration Attributes	2-9

CONTENTS

1	2.6.1 Simple Attributes	2-9
2	2.6.2 Complex Attributes	2-10
3	2.6.2.1 FTCReducedStrengthCipherringKey Attribute	2-10
4	2.6.2.2 RTCReducedStrengthCipherringKey Attribute	2-11
5	2.7 Non-Attribute Data	2-12
6	2.8 Protocol Numeric Constants	2-12
7	2.9 Session State Information	2-12
8	3 Basic Message Integrity Protocol	3-1
9	3.1 Overview	3-1
10	3.2 Primitives and Public Data	3-1
11	3.2.1 Commands	3-1
12	3.2.2 Return Indications	3-1
13	3.2.3 Procedure Calls	3-1
14	3.2.4 Local Common Data	3-2
15	3.2.5 Public Data	3-2
16	3.3 Protocol Data Unit	3-2
17	3.4 Protocol Initialization	3-2
18	3.4.1 Protocol Initialization for the InConfiguration Protocol Instance	3-2
19	3.4.2 Protocol Initialization for the InUse Protocol Instance	3-3
20	3.5 Procedures and Messages for the InConfiguration Instance of the Protocol	3-3
21	3.5.1 Procedures	3-3
22	3.5.2 Message Formats	3-3
23	3.6 Procedures and Messages for the InUse Instance of the Protocol	3-3
24	3.6.1 Procedures	3-3
25	3.6.1.1 Hard Commit Procedures	3-3
26	3.6.1.2 Soft Commit Procedures	3-3
27	3.6.1.3 Constructing the Message Integrity Key	3-3
28	3.6.1.4 Constructing the Cryptosync	3-4
29	3.6.1.5 Authentication Header	3-5
30	3.6.1.6 AUTHENTICATE_ADD_TAG procedures	3-6
31	3.6.1.7 AUTHENTICATE_CHECK_TAG procedures	3-8
32	3.6.2 Message Formats	3-9

CONTENTS

1	3.7 Interface to Other Protocols	3-9
2	3.7.1 Commands	3-9
3	3.7.2 Indications	3-9
4	3.8 Configuration Attributes	3-9
5	3.9 Non-Attribute Data	3-9
6	3.10 Protocol Numeric Constants	3-10
7	3.11 Session State Information	3-10
8	4 Basic Key Exchange Protocol	4-1
9	4.1 Overview	4-1
10	4.2 Primitives and Public Data	4-1
11	4.2.1 Commands	4-1
12	4.2.2 Return Indications	4-1
13	4.2.3 Local Common Data	4-1
14	4.2.4 Public Data	4-1
15	4.2.5 Interface to Other Protocols	4-2
16	4.2.5.1 Commands	4-2
17	4.2.5.2 Indications	4-2
18	4.3 Protocol Data Unit	4-2
19	4.4 Protocol Initialization for the InConfiguration Protocol Instance	4-2
20	4.5 Protocol Initialization for the InUse Protocol Instance	4-2
21	4.6 Procedures and Messages for the InConfiguration Instance of the Protocol	4-2
22	4.6.1 Procedures	4-2
23	4.6.2 Message Formats	4-3
24	4.7 Procedures and Messages for the InUse Instance of the Protocol	4-3
25	4.7.1 Procedures	4-3
26	4.7.1.1 Hard Commit Procedures	4-3
27	4.7.1.2 Soft Commit Procedures	4-3
28	4.7.1.3 Access Terminal Requirements	4-3
29	4.7.1.3.1 Initiating the key exchange	4-3
30	4.7.1.3.2 Processing a KeyResponse message	4-3
31	4.7.1.3.3 Processing a KeyReject message	4-4
32	4.7.1.4 Access Network Requirements	4-4

CONTENTS

1 4.7.1.4.1 Initiating the key exchange 4-4

2 4.7.1.4.2 Processing a KeyRequest message 4-4

3 4.7.1.4.3 Processing a KeyComplete message 4-5

4 4.7.1.5 MICKey Derivation..... 4-5

5 4.7.1.6 Message Integrity Key and Ciphering Key Generation 4-5

6 4.7.1.6.1 Temporary Security Key Derivation 4-5

7 4.7.1.6.2 Message Integrity Key and Ciphering Keys Generation from TSKey 4-6

8 4.7.1.7 EHMACH-SHA256(key, message, MAC_length)..... 4-6

9 4.7.2 Message Formats 4-6

10 4.7.2.1 KeyRequest..... 4-6

11 4.7.2.2 KeyResponse 4-7

12 4.7.2.3 KeyComplete 4-8

13 4.7.2.4 KeyReject..... 4-10

14 4.7.2.5 InitiateKeyRequest..... 4-10

15 4.7.3 Interface to Other Protocols 4-10

16 4.7.3.1 Commands 4-10

17 4.7.3.2 Indications 4-11

18 4.8 Configuration Attributes 4-11

19 4.9 Non-Attribute Data 4-11

20 4.10 Protocol Numeric Constants 4-11

21 4.11 Session State Information 4-11

22 4.11.1 PMK Parameter 4-11

23

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. AuthKeyIndex encoding3-5
5 Table 4-1. Definition of Result field4-9
6 Table 4-2. The Format of the Parameter Record for the PMK Parameter.....4-11
7

TABLES

- 1 No text.

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

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

- Overview for Ultra Mobile Broadband (UMB) Air Interface Specification
- Physical Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- MAC Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- Radio Link Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- Application Layer for Ultra Mobile Broadband (UMB) Air Interface Specification
- Connection Control Plane for Ultra Mobile Broadband (UMB) Air Interface Specification
- Session Control Plane for Ultra Mobile Broadband (UMB) Air Interface Specification
- Route Control Plane for Ultra Mobile Broadband (UMB) Air Interface Specification
- Broadcast-Multicast Upper Layers for Ultra Mobile Broadband (UMB) Air Interface Specification

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

This standard provides a specification for land mobile wireless systems based upon cellular principles. This Standard is one part of the IMT-2000 CDMA Multi-Carrier, IMT-2000 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."

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.

2 AES CIPHERING PROTOCOL

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

2.1 Primitives and Public Data

2.1.1 Commands

This protocol does not define any commands.

2.1.2 Return Indications

This protocol does not return any indications.

2.1.3 Procedure Calls

- ENCRYPT

- Inputs: *Direction*, *DataUnit*, *StreamID*, *RouteCounter*, *SARResetCounter*, *SARSequenceNumber*, *SARSequenceRolloverCounter*, *PayloadSize*, *Payload*

- Outputs: *Payload* (Encrypted), *CipheringKeyIndex*

- Possible values of each input and output are as follows:

- + *Direction* – ‘0’ (ForwardLink) or ‘1’ (ReverseLink)

- + *DataUnit* – ‘00’ (Octets), ‘01’ (RLP Packet Payload)

- + *StreamID* – 16-bit hexadecimal number

- + *RouteCounter* – 8-bit hexadecimal number

- + *SARResetCounter* – 8-bit hexadecimal number

- + *SARSequenceNumber* – Hexadecimal number of n bits, where n is less than or equal to 48.

- + *SARSequenceRolloverCounter* – Hexadecimal number of (48 – length of *SARSequenceNumber*) bits.

- + *PayloadSize* – Payload size in octets

- + *Payload* – Payload to be encrypted

- + *Payload* (Encrypted) – Encrypted payload of same size as input *Payload* and is available at same memory space as the input *Payload*.

- + *CipheringKeyIndex* – ‘00’ (not encrypted); ‘01’, ‘10’, ‘11’ (encrypted with key corresponding to index ‘01’, ‘10’ or ‘11’ respectively)

- DECRYPT

- Inputs: *Direction*, *DataUnit*, *StreamID*, *RouteCounter*, *SARResetCounter*, *SARSequenceNumber*, *SARSequenceRolloverCounter*, *PayloadSize*, *Payload*, *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* – 8-bit hexadecimal number
- 7 + *SARResetCounter* – 8-bit hexadecimal number
- 8 + *SARSequenceNumber* – Hexadecimal number of n bits, where n is less than
- 9 or 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
- 17 is 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 • Subtype for this protocol
- 22 • All data defined as Static Attribute, Static Non-Attribute Data, and Local Common
- 23 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 Protocol Initialization**

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 [8].

31 2.3.2 Protocol Initialization for the InUse Protocol Instance

32 Upon creation, the InUse instance of this protocol in the access terminal and access

33 network shall perform the procedures specified in [8].

2.4 Procedures and Messages for the InConfiguration Instance of the Protocol

2.4.1 Procedures

This protocol uses the services of the Session Configuration Protocol to perform negotiation of attribute values.

2.4.2 Message Formats

This protocol does not define any messages.

2.5 Procedures and Messages for the InUse Instance of the Protocol

2.5.1 Procedures

2.5.1.1 Hard Commit Procedures

The access terminal and the access network shall perform the procedures specified in [8] when directed by the InUse instance of the Session Configuration Protocol to execute the Hard Commit procedures.

2.5.1.2 Soft Commit Procedures

The access terminal and the access network shall perform the procedures specified in [8] when directed by the InUse instance of the Session Configuration Protocol to execute the Soft Commit procedures.

2.5.1.3 Constructing the Ciphering Key

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

- If the value of the FTCCiphering attribute is equal to 0x01, then the protocol shall construct the Ciphering key for the Forward Traffic Channel, FTCCipheringKey, as follows:
 - If the FACCipheringKey[KeyIndex] public data of the Key Exchange Protocol is set to NULL, the protocol shall set FTCCipheringKey to NULL.
 - Otherwise, the protocol shall perform the following:
 - + If the length of FACCipheringKey[KeyIndex] is equal to 128, then FTCCipheringKey shall be set to FACCipheringKey[KeyIndex].
 - + Otherwise, if the length of FACCipheringKey[KeyIndex] is greater than 128, then FTCCipheringKey shall be the 128 most significant bits of FACCipheringKey[KeyIndex].
 - + Otherwise, if the length of FACCipheringKey[KeyIndex] is less than 128, then FTCCipheringKey shall be the concatenation of zeros at the end (LSB) of FACCipheringKey[KeyIndex], such that the length of the result is 128.
 - The protocol shall perform the following:

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

34 2.5.1.4 Constructing the Cryptosync

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

Table 2-1. Subfield of the Cryptosync

Subfield	Length (bits)
FunctionCode	2
Reserved	13
Direction	1
RouteCounter	8
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 Link then
5 this field shall be set to '0'. Otherwise, this field shall be set to '1'.
6 Direction is received as *Direction* input to ENCRYPT and DECRYPT
7 procedure calls.
- 8 RouteCounter This field shall be set to the RouteCounter corresponding to the
9 payload being encrypted or decrypted. RouteCounter is received as
10 *RouteCounter* input to ENCRYPT and DECRYPT procedure calls.
- 11 StreamID This field shall be set to the StreamID corresponding to the payload
12 being encrypted or decrypted. StreamID is received as *StreamID*
13 input to ENCRYPT and DECRYPT procedure calls.
- 14 SARResetCounter This field shall be set to SARResetCounter corresponding to the
15 payload being encrypted or decrypted. SARResetCounter is received
16 as *SARResetCounter* input to ENCRYPT and DECRYPT procedure
17 calls.
- 18 VirtualSARSequenceNumber
19 This field shall be set to VirtualSARSequenceNumber corresponding
20 to the payload being encrypted or decrypted.
21 VirtualSARSequenceNumber value is set as specified in sections
22 2.5.1.5 and 2.5.1.6.

23 2.5.1.5 Encrypt Procedures

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

25 If any of the following conditions is true:

- 1 • The Cipherng attribute for the channel under consideration (e.g., FTCCipherng) is
2 equal to 0x00.
- 3 • The KeyIndex public data of the Key Exchange Protocol is set to NULL.
- 4 • The CipherngKey for the channel under consideration (e.g., FTCCipherngKey),
5 constructed as specified in 2.5.1.3 where KeyIndex is set to KeyIndex public data of
6 the Key Exchange Protocol, is NULL.

7 then, the ENCRYPT procedure shall set the outputs of the ENCRYPT procedure as follows:

- 8 • Set *Payload* to the input *Payload*.
- 9 • Set *CipherngKeyIndex* to '00'.

10 Otherwise, the ENCRYPT procedure shall perform the following:

- 11 • If *DataUnit* is set to '01' then the ENCRYPT procedure shall perform the following:
 - 12 – The ENCRYPT procedure shall call the ESP_AES procedure specified in [12] with
13 its inputs set as follows:
 - 14 + Set the *key* to the CipherngKey for the channel under consideration (e.g.,
15 FTCCipherngKey) constructed as specified in 2.5.1.3 where KeyIndex is set
16 to KeyIndex public data of the Key Exchange Protocol.
 - 17 + Set *fresh* to the value of the Cryptosync constructed as specified in 2.5.1.4,
18 where *VirtualSARSequenceNumber* is set to (*SARSequenceRolloverCounter* |
19 *SARSequenceNumber*).
 - 20 + Set the *freshsize* to the length of the Cryptosync in octets.
 - 21 + Set the *buf* to the address of the beginning of the memory space that
22 contains the *Payload*.
 - 23 + Set the *bit_offset* to zero.
 - 24 + Set the *bit_count* to 8 times *PayloadSize*.
 - 25 – After the ESP_AES procedure is returned, the ENCRYPT procedure shall set the
26 outputs of the ENCRYPT procedure as follows:
 - 27 + Set *Payload* to the output of the ESP_AES procedure which starts at the
28 memory space specified by *buf* and is of the same size as the input *Payload*.
 - 29 + Set *CipherngKeyIndex* to KeyIndex public data of the Key Exchange
30 Protocol.
- 31 • If *DataUnit* is set to '00' then the ENCRYPT procedure shall perform the following:
 - 32 – Set *VirtualSARSequenceNumberInUse* to (*SARSequenceRolloverCounter* |
33 *SARSequenceNumber*)
 - 34 – Set *n* to 1.

- 1 – The ENCRYPT procedure shall perform following steps *PayloadSize* times³:
- 2 + Call the ESP_AES procedure specified in [12] with its inputs set as follows:
- 3 Set the *key* to the CipheringKey for the channel under consideration (e.g.,
- 4 FTCCipheringKey) constructed as specified in 2.5.1.3 where KeyIndex is
- 5 set to KeyIndex public data of the Key Exchange Protocol.
- 6 Set *fresh* to the value of the Cryptosync constructed as specified in 2.5.1.4,
- 7 where VirtualSARSequenceNumber is set to $16 \times \lfloor$
- 8 VirtualSARSequenceNumberInUse / 16 \rfloor .
- 9 Set the *freshsize* to the length of the Cryptosync in octets.
- 10 Create a *temp_buf* and initialize it to (VirtualSARSequenceNumberInUse mod
- 11 16) octets of 0x00 followed by the nth octet of *Payload*. Set the *buf* to the
- 12 address of the beginning of the memory space that contains the *temp_buf*.
- 13 Set the *bit_offset* to zero.
- 14 Set the *bit_count* to $[(\text{VirtualSARSequenceNumberInUse mod } 16) + 1] \times 8$.
- 15 + After the ESP_AES procedure is returned, the ENCRYPT procedure shall skip
- 16 first (VirtualSARSequenceNumberInUse mod 16) octets of the output of the
- 17 ESP_AES procedure which starts at the memory space specified by
- 18 *temp_buf*, and overwrite nth octet of the *Payload* with the value of the next
- 19 octet of the ESP_AES procedure output.
- 20 + Increment value of n
- 21 + Increment value of VirtualSARSequenceNumberInUse modulo 2^{48} .
- 22 – The ENCRYPT procedure shall set the outputs of the ENCRYPT procedure as
- 23 follows:
- 24 + Set *CipheringKeyIndex* to KeyIndex public data of the Key Exchange
- 25 Protocol.

26 2.5.1.6 Decryption Procedures

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

28 If the Ciphering attribute for the channel under consideration (e.g., FTCCiphering) is equal

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

30 following:

- 31 • If *DataUnit* is set to '01' then the DECRYPT procedure shall perform the following:
- 32 – The DECRYPT procedure shall call the ESP_AES procedure specified in [12] with
- 33 its inputs set as follows:

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 + Set the *key* to the CipheringKey for the channel under consideration (e.g.,
2 FTCCipheringKey) constructed as specified in 2.5.1.3 where KeyIndex is set
3 to *CipheringKeyIndex*.
- 4 + Set *fresh* to the value of the Cryptosync constructed as specified in 2.5.1.4,
5 where VirtualSARSequenceNumber is set to $(SARSequenceRolloverCounter \mid$
6 $SARSequenceNumber)$.
- 7 + Set the *freshsize* to the length of the Cryptosync in octets.
- 8 + Set the *buf* to the address of the beginning of the memory space that
9 contains the *Payload*.
- 10 + Set the *bit_offset* to zero.
- 11 + Set the *bit_count* to 8 times *PayloadSize*.
- 12 – After the ESP_AES procedure is returned, the DECRYPT procedure shall set the
13 outputs of the DECRYPT procedure as follows:
- 14 + Set *Payload* to the output of the ESP_AES procedure which starts at the
15 memory space specified by *buf* and is of the same size as the input *Payload*.
- 16 • If *DataUnit* is set to '00' then the DECRYPT procedure shall perform the following:
- 17 – Set VirtualSARSequenceNumberInUse to $(SARSequenceRolloverCounter \mid$
18 $SARSequenceNumber)$.
- 19 – Set *n* to 1.
- 20 – The DECRYPT procedure shall perform following steps *PayloadSize* times⁴:
- 21 + Call the ESP_AES procedure specified in [12] with its inputs set as follows:
- 22 Set the *key* to the CipheringKey for the channel under consideration (e.g.,
23 FTCCipheringKey) constructed as specified in 2.5.1.3 where KeyIndex is
24 set to *CipheringKeyIndex*.
- 25 Set *fresh* to the value of the Cryptosync constructed as specified in 2.5.1.4,
26 where VirtualSARSequenceNumber is set to $16 \times \lfloor$
27 $VirtualSARSequenceNumberInUse / 16 \rfloor$.
- 28 Set the *freshsize* to the length of the Cryptosync in octets.
- 29 Create a *temp_buf* and initialize it to (VirtualSARSequenceNumberInUse mod
30 16) octets of 0x00 followed by the *n*th octet of *Payload*. Set the *buf* to the
31 address of the beginning of the memory space that contains the *temp_buf*.
- 32 Set the *bit_offset* to zero.
- 33 Set the *bit_count* to $\lfloor (VirtualSARSequenceNumberInUse \bmod 16) + 1 \rfloor \times 8$.

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.

Attribute ID	Attribute	Commit/ Scope	Values	Meaning
0x01	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.
			0x01	RLP packets destined for the RTC shall be encrypted by the sender and shall be decrypted by the receiver.
			0x02-0xff	Reserved

1 2.6.2 Complex Attributes

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

4 2.6.2.1 FTCRducedStrengthCipheringKey Attribute

5

Field	Length (bits)	Default Value
Length	16	N/A
AttributeID	8	N/A

One or more of the following record:

ValueID	8	N/A
FTCSaltLength	8	0
FTCSalt	FTCSaltLength × 8	N/A
FTCKeyEntropy	8	16

6 Length Length of the complex attribute in octets. The sender shall set this
7 field to the length of the complex attribute excluding the Length field.

8 AttributeID The sender shall set this field to 0x04.

9 ValueID This field identifies this particular set of values for the attribute.

10 FTCSaltLength The sender shall set this field to the length of the FTCSalt field in
11 octets.

12 FTCSalt The sender shall set this field to the value of the *Salt* input parameter
13 that is to be used in the KeyStrengthRedAlg procedure specified in
14 [13] for the FTC Ciphering key.

15 FTCKeyEntropy The sender shall set this field to the value of the *KeyEntropy* input
16 parameter that is to be used in the KeyStrengthRedAlg procedure

1 specified in [13] for the FTC Cipherring key. The valid values for this
 2 field are 0 through 16, inclusive.

3

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

4

5 2.6.2.2 RTCReducedStrengthCipherringKey Attribute

6

Field	Length (bits)	Default Value
Length	16	N/A
AttributeID	8	N/A

One or more of the following record:

ValueID	8	N/A
RTCSaltLength	8	0
RTCSalt	RTCSaltLength × 8	N/A
RTCKeyEntropy	8	16

7 **Length** Length of the complex attribute in octets. The sender shall set this
 8 field to the length of the complex attribute excluding the Length field.

9 **AttributeID** The sender shall set this field to 0x05.

10 **ValueID** This field identifies this particular set of values for the attribute.

11 **RTCSaltLength** The sender shall set this field to the length of the RTCSalt field in
 12 octets.

13 **RTCSalt** The sender shall set this field to the value of the *Salt* input parameter
 14 that is to be used in the KeyStrengthRedAlg procedure specified in
 15 [13] for the RTC Cipherring key.

16 **RTCKeyEntropy** The sender shall set this field to the value of the *KeyEntropy* input
 17 parameter that is to be used in the KeyStrengthRedAlg procedure
 18 specified in [13] for the RTC Cipherring key. The valid values for this
 19 field are 0 through 16, inclusive.

20

Commit	Hard
---------------	------

Scope	Dynamic
--------------	---------

1 **2.7 Non-Attribute Data**

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

3 **2.8 Protocol Numeric Constants**

4

Constant	Meaning	Value
N _{CPT} type	Type field for this protocol	[1 1]
N _{CP} AES	Subtype field for this protocol	[1]

5 **2.9 Session State Information**

6 The Session State Information record (see [8]) consists of parameter records.

7 The parameter records for this protocol consist of only the configuration attributes of this
8 protocol.

3 BASIC MESSAGE INTEGRITY PROTOCOL

3.1 Overview

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

3.2 Primitives and Public Data

3.2.1 Commands

This protocol does not define any commands.

3.2.2 Return Indications

This protocol does not define any indications.

3.2.3 Procedure Calls

- AUTHENTICATE_ADD_TAG

Inputs: *Direction*, *StreamID*, *RouteCounter*, *SARResetCounter*, *SARSequenceNumber*,
SARSequenceRolloverCounter, *UseAuthCounter*, *AuthCounter*, *InputPayloadSize*,
InputPayload

Outputs: *OutputPayloadSize*, *OutputPayload*

Possible values of each input and output are as follows:

Direction – ‘0’ (ForwardLink) or ‘1’ (ReverseLink)

StreamID – 16-bit hexadecimal number

RouteCounter – 8-bit hexadecimal number

SARResetCounter – 8-bit hexadecimal number

SARSequenceNumber – Hexadecimal number of n bits, where n is less than or equal to 48

SARSequenceRolloverCounter – Hexadecimal number of (48 – length of *SARSequenceNumber*) bits

UseAuthCounter – ‘0’ (FALSE), ‘1’ (TRUE)

AuthCounter – 16-bit hexadecimal number

InputPayloadSize – Input payload size in octets

InputPayload – Input payload

OutputPayloadSize – Output payload size in octets

OutputPayload – Output payload

- AUTHENTICATE_CHECK_TAG

1 Inputs: *Direction*, *StreamID*, *RouteCounter*, *SARResetCounter*, *SARSequenceNumber*,
2 *SARSequenceRolloverCounter*, *InputPayloadSize*, *InputPayload*

3 Outputs: *TagMatched*, *OutputPayloadSize*, *OutputPayload*

4 Possible values of each input and output are as follows:

5 *Direction* – ‘0’ (ForwardLink) or ‘1’ (ReverseLink)

6 *StreamID* – 16-bit hexadecimal number

7 *RouteCounter* – 8-bit hexadecimal number

8 *SARResetCounter* – 8-bit hexadecimal number

9 *SARSequenceNumber* – Hexadecimal number of n bits, where n is less than or
10 equal to 48.

11 *SARSequenceRolloverCounter* – Hexadecimal number of (48 – length of
12 *SARSequenceNumber*) bits.

13 *InputPayloadSize* – Input payload size in octets

14 *InputPayload* – Input payload

15 *TagMatched* – ‘1’ (TRUE), ‘0’ (FALSE)

16 *OutputPayloadSize* – Output payload size in octets

17 *OutputPayload* – Output payload

18 3.2.4 Local Common Data

19 This protocol does not define any Local Common Data.

20 3.2.5 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 **3.3 Protocol Data Unit**

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

27 **3.4 Protocol Initialization**

28 3.4.1 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 [8].

3.4.2 Protocol Initialization for the InUse Protocol Instance

Upon creation, the InUse instance of this protocol in the access terminal and access network shall perform the procedures specified in [8].

3.5 Procedures and Messages for the InConfiguration Instance of the Protocol

3.5.1 Procedures

This protocol uses the services of the Session Configuration Protocol to perform negotiation of attribute values.

3.5.2 Message Formats

This protocol does not define any messages.

3.6 Procedures and Messages for the InUse Instance of the Protocol

3.6.1 Procedures

3.6.1.1 Hard Commit Procedures

The access terminal and the access network shall perform the procedures specified in [8] when directed by the InUse instance of the Session Configuration Protocol to execute the Hard Commit procedures.

3.6.1.2 Soft Commit Procedures

The access terminal and the access network shall perform the procedures specified in [8] when directed by the InUse instance of the Session Configuration Protocol to execute the Soft Commit procedures.

3.6.1.3 Constructing the Message Integrity Key

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

- The protocol shall construct the Message Integrity key for the Forward Traffic Channel, FTCTMIKey as follows:

If the FACMIKey[KeyIndex] public data of the Key Exchange Protocol is set to NULL, the protocol shall set FTCTMIKey to NULL.

Otherwise, the protocol shall perform the following:

If the length of FACMIKey[KeyIndex] is equal to 128 bits, then FTCTMIKey shall be set to FACMIKey[KeyIndex].

Otherwise, if the length of FACMIKey[KeyIndex] is greater than 128 bits, then FTCTMIKey shall be the 128 most significant bits of FACMIKey[KeyIndex].

Otherwise, if the length of FACMIKey[KeyIndex] is less than 128 bits, then FTCTMIKey shall be the concatenation of zeros at the end (LSB) of FACMIKey[KeyIndex], such that the length of the result is 128 bits.

- The protocol shall construct the Message Integrity key for the Reverse Traffic Channel, RTCMIKey as follows:

If the RACMIKey[KeyIndex] public data of the Key Exchange Protocol is set to NULL, the protocol shall set RTCMIKey to NULL.

Otherwise, the protocol shall perform the following:

If the length of RACMIKey[KeyIndex] is equal to 128 bits, then RTCMIKey shall be set to RACMIKey[KeyIndex].

Otherwise, if the length of RACMIKey[KeyIndex] is greater than 128 bits, then RTCMIKey shall be the 128 most significant bits of RACMIKey[KeyIndex].

Otherwise, if the length of RACMIKey[KeyIndex] is less than 128 bits, then RTCMIKey shall be the concatenation of zeros at the end (LSB) of RACMIKey[KeyIndex], such that the length of the result is 128 bits.

3.6.1.4 Constructing the Cryptosync

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

Table 3-1. Subfield of the Cryptosync

Subfield	Length (bits)
Reserved	7
Direction	1
RouteCounter	8
StreamID	16
SARResetCounter	8
VirtualSARSequenceNumber	48

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

Direction If the payload is for Forward Link then this field shall be set to '0'. Otherwise, this field shall be set to '1'. Direction is received as *Direction* input to AUTHENTICATE_ADD_TAG and AUTHENTICATE_CHECK_TAG procedure calls.

RouteCounter This field shall be set to the RouteCounter corresponding to the payload. RouteCounter is received as *RouteCounter* input to AUTHENTICATE_ADD_TAG and AUTHENTICATE_CHECK_TAG procedure calls.

StreamID This field shall be set to the StreamID corresponding to the payload. StreamID is received as *StreamID* input to AUTHENTICATE_ADD_TAG and AUTHENTICATE_CHECK_TAG procedure calls.

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

5 VirtualSARSequenceNumber
 6 This field shall be set to VirtualSARSequenceNumber corresponding
 7 to the payload. VirtualSARSequenceNumber value is set as specified
 8 in sections 3.6.1.6 and 3.6.1.7.

9 3.6.1.5 Authentication Header

10 The Authentication Header, which precedes the payload, has the following format:
 11

Field	Length (bits)
MoreHeader	1
AuthKeyIndex	2
RouteIDIncluded	1
RouteID	0 or 4
AuthCounter	0 or 16
Reserved	0 or 4
AuthTag	0 or 64

12 MoreHeader The protocol shall set this field to '1' if there is another
 13 Authentication Header following this Authentication Header;
 14 otherwise, the sender shall set this field to '0'.

15 AuthKeyIndex The protocol shall set this field to indicate index of the key used for
 16 authentication of the payload as specified in Table 3-2.

17 **Table 3-2. 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'

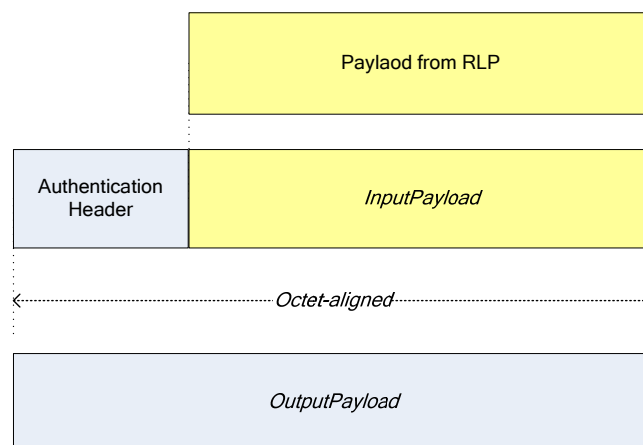
18 RouteIDIncluded The protocol shall set this field to '1' if the RouteID field is included;
 19 otherwise, the protocol shall set this field to '0'.

1	RouteID	The protocol shall omit this field if the RouteIDIncluded field is set to
2		'0'; otherwise, the protocol shall include this field and set it to the
3		RouteID assigned to the Route for which the AuthTag is included.
4	AuthCounter	The protocol shall omit this field if the RouteIDIncluded field is set to
5		'0' or AuthKeyIndex is set to '00'; otherwise, the protocol shall
6		include this field and set it to the AuthCounter to be included in
7		Cryptosync.
8	Reserved	The protocol shall omit this field if the RouteIDIncluded field is set to
9		'1'; otherwise, the protocol shall include this field and set all the bits
10		in this field to '0'.
11	AuthTag	The protocol shall omit this field if the AuthKeyIndex field is set to
12		'00'; otherwise, the protocol shall include this field and set it to
13		authentication tag.

14 3.6.1.6 AUTHENTICATE_ADD_TAG procedures

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

18



19

20 **Figure 3-1. AUTHENTICATE_ADD_TAG procedure call payloads**

21 If any of the following conditions is true:

- 22 • The KeyIndex public data of the Key Exchange Protocol is set to NULL.
- 23 • The MIKey for the channel under consideration (e.g., FTCMIKey), constructed as
 24 specified in 3.6.1.3 where KeyIndex is set to KeyIndex public data of the Key Exchange
 25 Protocol, is NULL.
- 26 • Payload contains a message for which authentication tag is not required (See [1] and
 27 each protocol text) and if protocol decides not to include the authentication tag.

1 then, the AUTHENTICATE_ADD_TAG procedure shall set the outputs of the
2 AUTHENTICATE_ADD_TAG procedure as follows:

- 3 • Construct Authentication Header with the AuthKeyIndex field set to '00'.
- 4 • Set *OutputPayload* to (Authentication Header | *InputPayload*).
- 5 • Set *OutputPayloadSize* to (*InputPayloadSize* + size of the Authentication Header in
6 octets).

7 Otherwise, the AUTHENTICATE_ADD_TAG procedure shall perform the following:

- 8 • The AUTHENTICATE_ADD_TAG procedure shall call the ENC_CMACE procedure
9 specified in [13] with its inputs set as follows:

10 Set the *key* to the MIKey for the channel under consideration (e.g., FTCEMIKey)
11 constructed as specified in 3.6.1.3 where KeyIndex is set to KeyIndex public
12 data of the Key Exchange Protocol.

13 Set *fresh* to the value of the Cryptosync constructed as specified in 3.6.1.4 with
14 VirtualSARSequenceNumber set as follows:

15 If *UseAuthCounter* is TRUE, then VirtualSARSequenceNumber is set to (32 bits of
16 '0' | *AuthCounter*). Otherwise, VirtualSARSequenceNumber is set to
17 (*SARSequenceRolloverCounter* | *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 the
20 *InputPayload*.

21 Set the *bit_offset* to zero.

22 Set the *bit_count* to 8 times *InputPayloadSize*.

23 Set the *MAC_length* to 8

- 24 • After the ENC_CMACE procedure is returned, the AUTHENTICATE_ADD_TAG procedure
25 shall set the outputs of the AUTHENTICATE_ADD_TAG procedure as follows:

26 Construct Authentication Header with fields set as follows:

27 Set the AuthTag field to *MAC_result* output of CMACE procedure.

28 Set the AuthKeyIndex field set to KeyIndex public data of the Key Exchange
29 Protocol.

30 If *UseAuthCounter* is TRUE, then set the AuthCounter field to *AuthCounter*.
31 Otherwise, omit the AuthCounter field.

32 Set *OutputPayload* to (Authentication Header | *InputPayload*)

33 Set *OutputPayloadSize* to (*InputPayloadSize* + size of the Authentication Header in
34 octets)

3.6.1.7 AUTHENTICATE_CHECK_TAG procedures

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

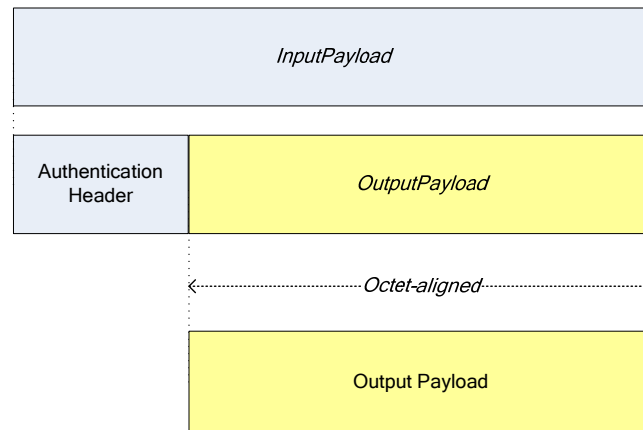


Figure 3-2. AUTHENTICATE_CHECK_TAG procedure call payloads

The AUTHENTICATE_CHECK_TAG procedure shall perform the following:

- The *InputPayload* consists of (Authentication Header | *OutputPayload*).
- The AUTHENTICATE_CHECK_TAG procedure shall select an Authentication Header from *InputPayload* as specified below. The AUTHENTICATE_CHECK_TAG procedure shall use field values from the selected header to perform rest of the operations.

If only one Authentication Header is included, then select that Authentication Header.

Otherwise, select the Authentication Header with RouteID field that matches with the RouteID of this protocol stack.

- The AUTHENTICATE_CHECK_TAG procedure shall remove Authentication Headers from *InputPayload* to produce *OutputPayload*.
- If the AuthKeyIndex field of the Authentication Header is set to '00', then AUTHENTICATE_CHECK_TAG procedure shall set the outputs of the AUTHENTICATE_CHECK_TAG procedure as follows::

Set *TagMatched* to TRUE.

Set *OutputPayloadSize* to size of *OutputPayload* in octets.

- Otherwise, the AUTHENTICATE_CHECK_TAG procedure shall call the ENC_CMACE procedure specified in [13] with its inputs set as follows:

- 1 Set the *key* to the MIKey for the channel under consideration (e.g., FTCKMIKey)
2 constructed as specified in 3.6.1.3 where KeyIndex is set to AuthKeyIndex field
3 of the Authentication Header.
- 4 Set *fresh* to the value of the Cryptosync constructed as specified in 3.6.1.4 with
5 VirtualSARSequenceNumber set as follows:
- 6 If the AuthCounter field is included in the Authentication Header, then
7 VirtualSARSequenceNumber is set to (32 bits of '0' | AuthCounter field in the
8 Authentication Header). Otherwise, VirtualSARSequenceNumber is set to
9 (*SARSequenceRolloverCounter* | *SARSequenceNumber*).
- 10 Set the *freshsize* to the length of the Cryptosync in octets.
- 11 Set the *buf* to the address of the beginning of the memory space that contains the
12 *OutputPayload*.
- 13 Set the *bit_offset* to zero.
- 14 Set the *bit_count* to 8 times size of *OutputPayload* in octets.
- 15 Set the *MAC_length* to 8
- 16 • After the ENC_CMACE procedure is returned, the AUTHENTICATE_CHECK_TAG
17 procedure shall set the outputs of the AUTHENTICATE_CHECK_TAG procedure as
18 follows:
- 19 If *MAC_result* output of ENC_CMACE procedure matches the AuthTag field of the
20 Authentication Header, then set *TagMatched* to TRUE. Otherwise, set
21 *TagMatched* to FALSE.
- 22 Set *OutputPayloadSize* to size of *OutputPayload* in octets.

23 3.6.2 Message Formats

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

25 **3.7 Interface to Other Protocols**

26 3.7.1 Commands

27 This protocol does not issue any commands.

28 3.7.2 Indications

29 This protocol does not register to receive any indications.

30 **3.8 Configuration Attributes**

31 This protocol does not define any attributes.

32 **3.9 Non-Attribute Data**

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

3.10 Protocol Numeric Constants

Constant	Meaning	Value
N _{MIPType}	Type field for this protocol	[11]
N _{BMIP}	Subtype field for this protocol	[1]

3.11 Session State Information

The Session State Information record (see [8]) consists of parameter records.

The parameter records for this protocol consist of only the configuration attributes of this protocol.

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
8 Master 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 does not define any Local Common Data.

19 4.2.4 Public Data

20 This protocol shall make the following data public:

- 21 • Subtype for this protocol
- 22 • PMKList[] (Access Terminal only)
23 A list of all Pairwise Master Keys established by this route.
- 24 • KeyIndex
25 The key index in use. Valid values are '01', '10' and '11'.
- 26 • FACMIKey[i] and its length for values of *i* '01' through '11'
27 The Message Integrity key for use on Forward Assigned Channels (e.g., the Forward
28 Traffic Channel).
- 29 • RACMIKey[i] and its length for values of *i* '01' through '11'
30 The Message Integrity key for use on Reverse Assigned Channels (e.g., the Reverse
31 Traffic Channel).

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

8 4.2.5 Interface to Other Protocols

9 4.2.5.1 Commands

10 This protocol does not define any commands.

11 4.2.5.2 Indications

12 This protocol does not register to receive any indications.

13 **4.3 Protocol Data Unit**

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

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

17 **4.4 Protocol Initialization for the InConfiguration Protocol Instance**

18 Upon creation, the InConfiguration instance of this protocol in the access terminal and the
19 access network shall perform the procedures specified in [8].

20 **4.5 Protocol Initialization for the InUse Protocol Instance**

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

- 23 • Perform the procedures specified in [8]
- 24 • Set KeyIndex to NULL.
- 25 • Set FACMIKey[i] to NULL, for values of *i* '01' through '11'.
- 26 • Set RACMIKey[i] to NULL, for values of *i* '01' through '11'.
- 27 • Set FACCIpheringKey[i] to NULL, for values of *i* '01' through '11'.
- 28 • Set RACCIpheringKey[i] to NULL, for values of *i* '01' through '11'.

29 **4.6 Procedures and Messages for the InConfiguration Instance of the Protocol**

30 4.6.1 Procedures

31 This protocol uses the services of the Session Configuration Protocol to perform negotiation
32 of attribute values

1 4.6.2 Message Formats

2 This protocol does not define any messages.

3 **4.7 Procedures and Messages for the InUse Instance of the Protocol**

4 4.7.1 Procedures

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

8 4.7.1.1 Hard Commit Procedures

9 The access terminal and the access network shall perform the procedures specified in [8]
10 when directed by the InUse instance of the Session Configuration Protocol to execute the
11 Hard Commit procedures.

12 4.7.1.2 Soft Commit Procedures

13 The access terminal and the access network shall perform the procedures specified in [8]
14 when directed by the InUse instance of the Session Configuration Protocol to execute the
15 Soft Commit procedures.

16 4.7.1.3 Access Terminal Requirements

17 4.7.1.3.1 Initiating the key exchange

18 Access terminal may initiate the key exchange procedure by sending a KeyRequest
19 message.

20 Upon receiving the InitiateKeyRequest message, the access terminal shall send a
21 KeyRequest message unless access terminal has already sent a KeyRequest message and is
22 awaiting the response.

23 4.7.1.3.2 Processing a KeyResponse message

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

- 27 • The access terminal shall identify the PairwiseMasterKey from PMKList of any route
28 that satisfies $\text{PairwiseMasterKeyID} = \text{EHMAC-SHA256}(\text{key}=\text{PairwiseMasterKey},$
29 $\text{message} = \text{"PairwiseMasterKeyID"}, \text{MAC_length}=8)$, where PairwiseMasterKeyID is a
30 field of the received KeyResponse message, "PairwiseMasterKeyID" is the ASCII encoded
31 value of the string, and the EHMAC-SHA256 function is specified in 4.7.1.7.
- 32 • If the access terminal cannot identify a valid PairwiseMasterKey that satisfies the above
33 condition, then the access terminal shall send a KeyComplete message with Result set
34 to 0x03, declare failure, and stop performing the rest of the key exchange procedure.
- 35 • The access terminal shall generate MICKey as specified in 4.7.1.5.

- 1 • The access terminal shall generate a MessageIntegrityCode as EHMAC-
2 SHA256(key=MICKey, message=*Message*, MAC_length=16), where *Message* is the
3 received KeyResponse message with the MessageIntegrityCode field set to zero, and the
4 EHMAC-SHA256 function is specified in 4.7.1.7.
- 5 • If the MessageIntegrityCode computed in the previous step does not match the
6 MessageIntegrityCode field of KeyResponse message, then the access terminal shall
7 declare failure, send a KeyComplete message with Result set to 0x01, and stop
8 performing the rest of the key exchange procedure.
- 9 • If the supported ProtocolSetIdentifiers sent by the access network in the KeyResponse
10 message do not match with the ProtocolSetIdentifiers supported by the access terminal
11 (i.e. either all the ProtocolSetIdentifiers supported by the access terminal are not
12 included in the KeyResponse message or KeyResponse message includes a
13 ProtocolSetIdentifiers that is not supported by the access terminal), then the access
14 terminal shall declare failure, send a KeyComplete message with Result set to 0x02,
15 and stop performing the rest of the key exchange procedure.
- 16 • If the first ProtocolSetIdentifier field sent by the access network in the KeyResponse
17 message do not match with the ProtocolSetIdentifier currently in use by the access
18 terminal, then the access terminal shall declare failure, send a KeyComplete message
19 with Result set to 0x04, and stop performing the rest of the key exchange procedure.
- 20 • The access terminal shall generate Message Integrity Keys and Ciphering Keys as
21 specified in 4.7.1.6 for KeyIndex value specified in the KeyResponse message. The
22 access terminal shall set KeyIndex public data to KeyIndex value specified in the
23 KeyResponse message.
- 24 • The access terminal shall send a KeyComplete message with Result set to 0x00.

25 4.7.1.3.3 Processing a KeyReject message

26 Upon receiving the KeyReject message with a TransactionID field that matches the
27 TransactionID field of the associated KeyRequest message, the access terminal shall
28 terminate the key exchange procedure.

29 4.7.1.4 Access Network Requirements

30 4.7.1.4.1 Initiating the key exchange

31 Access network may initiate the key exchange procedure by sending an InitiateKeyRequest
32 message.

33 4.7.1.4.2 Processing a KeyRequest message

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

- 36 • The access network shall send a KeyResponse message or a KeyReject message.
- 37 • If a KeyResponse message was sent, the access network shall generate MICKey as
38 specified in 4.7.1.5.

1 4.7.1.4.3 Processing a KeyComplete message

2 After receiving a KeyComplete message with a TransactionID field that matches the
3 TransactionID field of the associated KeyResponse message, the access network shall
4 perform the following:

- 5 • The access network shall generate a MessageIntegrityCode as EHMAL-
6 SHA256(key=MICKey, message=*Message*, MAC_length=16), where *Message* is the
7 received KeyComplete message with the MessageIntegrityCode field set to zero, and the
8 EHMAL-SHA256 function is specified in 4.7.1.7.
- 9 • If the MessageIntegrityCode computed in the previous step does not match the
10 MessageIntegrityCode field of KeyComplete message, then the access network shall
11 declare failure, and stop performing the rest of the key exchange procedures.
- 12 • If the Result field of the KeyComplete message is not 0x00, then the access network
13 shall declare failure and stop performing the rest of the key exchange procedure.
- 14 • The access network shall generate Message Integrity Keys and Ciphering Keys as
15 specified in 4.7.1.6 for KeyIndex value specified in the KeyResponse message. The
16 access network shall set KeyIndex public data to KeyIndex value specified in the
17 KeyResponse message.

18 4.7.1.5 MICKey Derivation

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

- 20 • Set MICKey to zero and its length to 128.
- 21 • Set MICKey to EHMAL-SHA256(key=PairwiseMasterKey, message=
22 "MICKey"|ATNonce|ANNonce, MAC_length=16), where "MICKey" is the ASCII encoded
23 value of the string, and the EHMAL-SHA256 function is specified in 4.7.1.7.

24 4.7.1.6 Message Integrity Key and Ciphering Key Generation

25 The access terminal and the access network shall generate a TSKey as specified in
26 4.7.1.6.1. The access terminal and the access network shall generate Message Integrity and
27 Ciphering keys from TSKey as specified in 4.7.1.6.2

28 4.7.1.6.1 Temporary Security Key Derivation

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

- 30 • Set TSKey to zero and its length to $N_{\text{BKEPTKeyLen}}$.
- 31 • Set j to an 8-bit number with value zero.
- 32 • while $j < N_{\text{BKEPTKeyLen}}/256$

- 1 – Set TSKey to $N_{\text{BKPE}} \times \text{TSKeyLen}$ least significant bits of { TSKey | EHMACHMAC-SHA256(key=PairwiseMasterKey, message= “TSK” | ATNonce | ANNonce | j, MAC_length=32) }, where “TSK” is the ASCII encoded value of the string, *j* is represented as an 8-bit field, and the EHMACHMAC-SHA256 function is specified in 4.7.1.7.
- 2
- 3
- 4
- 5
- 6 – Set *j* to *j*+1.

7 4.7.1.6.2 Message Integrity Key and Ciphering Keys Generation from TSKey

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

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

- 10
- 11
- 12 • The access network and the access terminal shall set FACMIKey[*i*] to TSKey[127:0].
 - 13 • The access network and the access terminal shall set RACMIKey[*i*] to TSKey[127:0].
 - 14 • The access network and the access terminal shall set FACCIpheringKey[*i*] to TSKey[255:128].
 - 15
 - 16 • The access network and the access terminal shall set RACCIpheringKey[*i*] to TSKey[255:128].
 - 17

18 4.7.1.7 EHMACHMAC-SHA256(key, message, MAC_length)

19 The EHMACHMAC-SHA256 procedure shall call ehmacsha256 procedure as specified in [13] with following inputs:

- 20
- 21 • Set the *key_length* to the length of the key in bits,
 - 22 • Set the *key* to the key,
 - 23 • Set the *message* to the message, and
 - 24 • Set the *message_length* to the length of the message in bits,
 - 25 • Set the *message_offset* to 0,
 - 26 • Set the *MAC_length* to $8 \times \text{MAC_length}$.

27 The output of the EHMACHMAC-SHA256 function shall be set to the output of the ehmacsha256 procedure.

28

29 4.7.2 Message Formats

30 4.7.2.1 KeyRequest

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

32

Field	Length (bits)
MessageID	8
TransactionID	8
ATNonce	128

- 1 MessageID The access terminal shall set this field to 0x00.
- 2 TransactionID The access terminal shall increment this value for each new
3 KeyRequest message sent.
- 4 ATNonce The access terminal shall set this field to a 128-bit random number.

5

Channels	RTC	RLP	Reliable
Addressing	unicast		

6 4.7.2.2 KeyResponse

- 7 The access network may send the KeyResponse message in response to the KeyRequest
8 message.

9

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
----------------------	-----

- 10 MessageID The access network shall set this field to 0x01.
- 11 TransactionID The access network shall set this field to the value of the
12 TransactionID field of the KeyRequest message to which the access
13 network is responding.

- 1 KeyIndex The access network shall set this field to key index for which this key
 2 exchange is being initiated. Valid values for this field are '01' through
 3 '11'. The access network shall not set this field to KeyIndex public
 4 data.
- 5 Reserved The access network shall set all the bits in this field to '0'. The access
 6 terminal shall ignore this field.
- 7 PairwiseMasterKeyID
 8 The access network shall set this field to EHMAL-
 9 SHA256(key=PairwiseMasterKey, message= "PairwiseMasterKeyID",
 10 MAC_length=8), where PairwiseMasterKey is a PairwiseMasterKey to
 11 be used, "PairwiseMasterKeyID" is the ASCII encoded value of the
 12 string, and the EHMAL-SHA256 function is specified in 4.7.1.7.
- 13 ANNonce The access network shall set this field to a 128-bit random number.
- 14 NumProtocolSetIdentifiers
 15 The access network shall set this field to the number of
 16 ProtocolSetIdentifiers supported by the access terminal.
 17 ProtocolSetIdentifiers supported by the access terminal are known to
 18 the access network through session information.
- 19 ProtocolSetIdentifier
 20 The access network shall set this field to the ProtocolSetIdentifier
 21 supported by the access terminal. The access network shall include
 22 the ProtocolSetIdentifier currently in use at the beginning. Protocol
 23 subtypes supported by the access terminal are known to the access
 24 network through session information.
- 25 MessageIntegrityCode
 26 The access network shall set this field to EHMAL-
 27 SHA256(key=MICKey, message=*Message*, MAC_length=16), where
 28 *Message* is set to all fields of this message with this field set to zero,
 29 and the EHMAL-SHA256 function is specified in 4.7.1.7.

Channels	FTC	RLP	Reliable
Addressing	unicast		

31 4.7.2.3 KeyComplete

32 The access terminal sends the KeyComplete message in response to the KeyResponse
 33 message.
 34

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

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

5 **Table 4-1. 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

- 6 MessageIntegrityCode
- 7 If Result is 0x00, then the access terminal shall set this field to
- 8 EHMACH-SHA256(key=MICKey, message=*Message*, MAC_length=16),
- 9 where *Message* is set to all fields of this message with this field set to
- 10 zero, and the EHMACH-SHA256 function is specified in 4.7.1.7.
- 11 Otherwise, the access terminal shall omit this field.
- 12

Channels	RTC	RLP	Reliable
Addressing	unicast		

1 4.7.2.4 KeyReject

2 The access network may send the KeyReject message in response to the KeyRequest
3 message.

4

Field	Length (bits)
MessageID	8
TransactionID	8

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

6 TransactionID The access network shall set this field to the value of the
7 TransactionID field of the KeyRequest message to which the access
8 network is responding.

9

Channels	FTC	RLP	Reliable
Addressing	unicast		

10 4.7.2.5 InitiateKeyRequest

11 The access network may send the InitiateKeyRequest message to request the access
12 terminal to send a KeyRequest message.

13

Field	Length (bits)
MessageID	8

14 MessageID The access network shall set this field to 0x04.

15

Channels	FTC	RLP	Reliable
Addressing	unicast		

16 4.7.3 Interface to Other Protocols

17 4.7.3.1 Commands

18 This protocol does not issue any commands.

1 4.7.3.2 Indications

2 This protocol does not register to receive any indications.

3 **4.8 Configuration Attributes**

4 This protocol does not define any attributes.

5 **4.9 Non-Attribute Data**

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

7 **4.10 Protocol Numeric Constants**

Constant	Meaning	Value
$N_{KEPT_{Type}}$	Type field for this protocol	[11]
N_{BKEP}	Subtype field for this protocol	[1]
$N_{BKEPTS_{KeyLen}}$	Length of Temporary Security Key in units of bits	256

8 **4.11 Session State Information**

9 The Session State Information record (see [8]) consists of parameter records.

10 This protocol defines the following parameter record in addition to the configuration
11 attributes for this protocol.

12 4.11.1 PMK Parameter

13 This is an optional parameter record. That is, this parameter record may be omitted from
14 the session.15 **Table 4-2. The Format of the Parameter Record for the PMK Parameter**

Field	Length (bits)
ParameterType	8
Length	8
PMKCount	8

PMKCount occurrences of the following four fields:

PMKLength	8
PMK	$PMKLength \times 8$
PMKGenerationTime	24
PMKMinLifeTime	24
PMKMaxLifeTime	24

16 ParameterType This field shall be set to 0x01 for this parameter record.

1	Length	This field shall be set to the length of this parameter record in units
2		of octets excluding the Length field.
3	PMKCount	This field shall be set to the number of occurrences of the PMK field
4		in this parameter record.
5	PMKLength	This field shall be set to the length of the PMK field in units of octets.
6	PMK	This field shall be set to a PairwiseMasterKey.
7	PMKGenerationTime	This field shall be set to $X \bmod 2^{24}$; where X is the CDMA system
8		time, in units of seconds, at which the PairwiseMasterKey was
9		generated.
10	PMKMinLifeTime	This field shall be set to minimum lifetime of the PairwiseMasterKey
11		in units of seconds.
12	PMKMaxLifeTime	This field shall be set to maximum lifetime of the PairwiseMasterKey
13		in units of seconds.