Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**
Title	**MBS AES-CTR Test Vector and Test Program Changes Rev 1**
Date Submitted	**2005-03-17**
Source(s)	JUNHYUK SONG, JICHEOL LEE, YONG CHANG Voice: +82-31-279-3639 Samsung Electronics junhyuk.song@samsung.com Zohar, ilan ilan.zohar@intel.com Intel
Re:	
Abstract	MBS AES-CTR Test Vector
Purpose	Per C802.16e-05/047r1, MBS AES-CTR Test Vector and figure change
Notice	This document has been prepared to assist IEEE 802.16. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.
Release	The contributor grants a free, irrevocable license to the IEEE to incorporate material contained in this contribution, and any modifications thereof, in the creation of an IEEE Standards publication; to copyright in the IEEE's name any IEEE Standards publication even though it may include portions of this contribution; and at the IEEE's sole discretion to permit others to reproduce in whole or in part the resulting IEEE Standards publication. The contributor also acknowledges and accepts that this contribution may be made public by IEEE 802.16.
Patent Policy and Procedures	The contributor is familiar with the IEEE 802.16 Patent Policy and Procedures <http://ieee802.org/16/ipr/patents/policy.html>, including the statement "IEEE standards may include the known use of patent(s), including patent applications, provided the IEEE receives assurance from the patent holder or applicant with respect to patents essential for compliance with both mandatory and optional portions of the standard." Early disclosure to the Working Group of patent information that might be relevant to the standard is essential to reduce the possibility for delays in the development process and increase the likelihood that the draft publication will be approved for publication. Please notify the Chair <mailto:chair@wirelessman.org> as early as possible, in written or electronic form, if patented technology (or technology under patent application) might be incorporated into a draft standard being developed within the IEEE 802.16 Working Group. The Chair will disclose this notification via the IEEE 802.16 web site <http://ieee802.org/16/ipr/patents/notices>.

PKM version 2

MBS AES-CTR Test Vector and Figure Changes

JUNHYUK SONG, JICHEOL LEE, YONG CHANG
Samsung Electronics

Zohar ilan
Intel

INTRODUCTION

Per decision of last meeting, Data encryption with AES in CTR mode is newly proposed and D6 text is changed.

According to newly proposed usage which make use of ROC(roll-over-counter) and Frame Number, there are two things that should be changed together like followings:

1. Figure 137a MBS MAC PDU Ciphertext payload format
2. MBS AES-CTR Test Vector and test program
3. Clarifying the text including note about ROC

Changes to 802.16e D6 text

7.8.4.1.1PDU payload format

Counter mode requires unique initial counter and key pair across all messages. This section describes the initialization of the 128-bit initial counter, constructed from the 24-bit PHY synchronization field or frame number and a new 8-bit Rollover counter (ROC).

(Note: When we start to deal with a new PDU we have a new frame number and therefore reinitialize the counter. When the frame number reaches 0x000000 (from 0xFFFFFF), we increment ROC)

The PDU payload for AES-CTR encryption shall be prepended with the 8-bit ROC , i.e., the ROC is the 8 MSBs of the 32 bit nonce.. The ROC shall be transmitted in little endian order. The ROC shall not be encrypted.

Any tuple value of {AES Counter, KEY} shall not be used more than once for the purposes of encrypting a block.. MS and BS shall ensure that a new MTEK is requested and transferred before the ROC reaches 0xFF. The 32bit nonce made out of ROC and 24bits frame number shall be repeated four times to construct 128-bit counter block required by the AES-128 cipher. (~~e.g~~. initial counter = NONCE|NONCE|NONCE|NONCE). This mechanism can reduce per-PDU overhead of transmitting the full counter. In other words, at the most 2^32 PDUs can be encrypted with a single MTK.

The plaintext PDU shall be encrypted using the active MBS_Traffic_key (MTK) derived from MAK and-MGTEK, according to CTR mode specification. A different 128-bit counter value is used to encrypt each 128-bit block within a PDU. ~~This can be achieved by only incrementing the lowest 32 bits by 1; the lowest 32 bit value may rotate to 0 and counted up.~~

The processing yields a payload that is 8 bits longer than the plaintext payload.

[Change Figure 137a.]

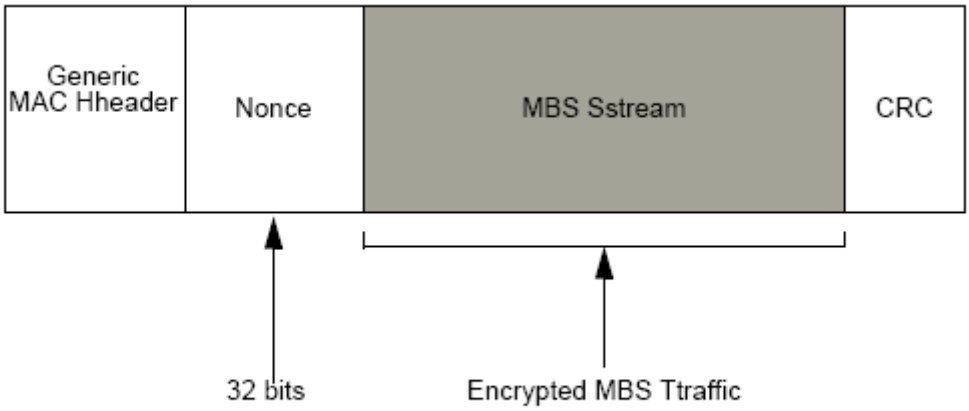
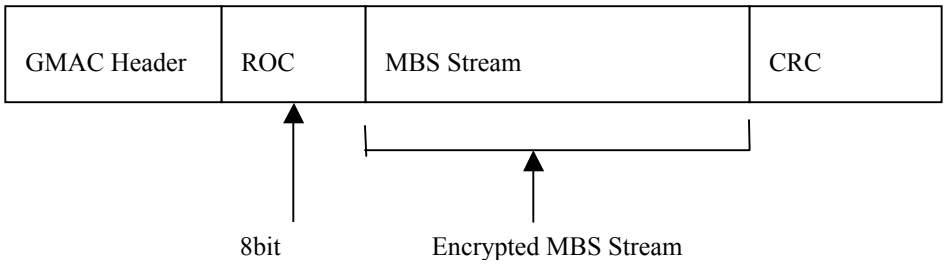


Figure 137a—MBS MAC PDU Ciphertext ~~P~~payload ~~F~~format



7a –MBS MAC PDU Ciphertext payload format

[Change E.1.1.1]

E.1.1.1 Test Vector

E.1.1.1.1 Test 1

PLAIN TEXT: 64 Bytes

d8 65 c9 cd ea 33 56 c5 48 8e 7b a1 5e 84 f4 eb
a3 b8 25 9c 05 3f 24 ce 29 67 22 1c 00 38 84 d7
9d 4c a4 87 7f fa 4b c6 87 c6 67 e5 49 5b cf ec
12 f4 87 17 32 aa e4 5a 11 06 76 11 3d f9 e7 da

Roll-over-counter: 1 Byte

00

PHY Syncronization: 3 Byte

ffffff

Counter: 16 Bytes

00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff

Key: 16Bytes

00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff

CIPHER TEXT: 64 Byte + 1 Byte(Roll-over-counter)

00 65 e1 16 74 b6 2e 38 bc ad 88 b4 d8 30 7e 44
2b cb 5d 66 ee 5c 1c 82 ca 3d cf 21 db 90 c9 13
b4 25 10 f4 d1 41 1e 04 8a 60 99 cf 02 32 d4 fe
24 db 28 78 f0 fb 1c b6 c8 b5 41 63 6d e9 a6 1b
15

DECRYPT TEXT: 64 Byte

d8 65 c9 cd ea 33 56 c5 48 8e 7b a1 5e 84 f4 eb
a3 b8 25 9c 05 3f 24 ce 29 67 22 1c 00 38 84 d7
9d 4c a4 87 7f fa 4b c6 87 c6 67 e5 49 5b cf ec

12 f4 87 17 32 aa e4 5a 11 06 76 11 3d f9 e7 da

E.1.1.1.2 Test 2

PLAIN TEXT: 256 Bytes

45 8b 61 c3 84 ab 89 0b 71 ef ef b9 49 be a4 5b
b1 2b 71 e2 d5 55 3b e5 5a b0 f5 97 a9 dc 71 ed
66 d1 b0 ea 7c 38 f4 ec 26 e2 a5 6f 9f 48 ca 4f
73 3a 31 47 8f 6b 2c e9 1b 21 7f c3 fd f0 b0 63
c0 5f 4c 3c 96 3f 28 bc 21 cc 2b bf 14 f4 0e 86
2e 3e cd bc a9 f8 a4 c3 18 23 86 15 12 35 77 d2
93 c2 0e 29 00 35 e4 21 00 0e df 13 02 ed 99 2f
2a 65 ea d2 5c 8e 95 74 b0 1a 88 c2 4e ff 94 e1
c0 a2 0a c0 d6 ed e0 d5 fb bf e8 fc ab 80 2a d5
e4 14 a7 40 a2 3b b4 52 55 3c 13 a3 3a a7 83 f9
48 8c b9 1d 79 98 f2 74 57 da 70 01 59 9a d6 3c
ad 7c 7c 4f b7 2f a0 0b 6a b3 ad a4 59 30 9c a1
bc 55 be 34 ec b0 a8 42 89 17 43 e1 b0 18 1d 5d
94 98 ab 4a c7 4a 55 31 fc 01 d4 55 31 70 f6 ec
c4 b3 20 b0 63 c7 f2 eb dd 35 cc 8d 4d e8 e9 e0
80 94 2a 47 de 7f 77 da 7f 4b 2f b0 bb 24 9b 7f

Roll-over-counter: 1 Byte

00

PHY Syncronization: 3 Byte

ffffff

Counter: 16 Bytes

00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff

Key: 16Bytes

00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff

CIPHER TEXT: 256 Byte + 1 Byte(Roll-over-counter)

00 f8 0f be 7a d8 b6 e7 72 94 e9 20 c0 27 44 14
9b d9 ce 32 90 8c 76 9d e1 4e 18 f6 50 39 2d e6

8e de 8d e0 bc 42 dc bb a0 c1 bd 0d 88 e4 c7 fb
87 ba e6 ce a0 46 dd 7e 7b bf 66 6a bf 29 af 4c
ac ec 7b ca 8c 91 41 41 f5 18 98 be 04 ec 83 7b
b3 1e 08 65 93 d9 74 fb c2 58 c4 d1 e9 17 fa a8
08 09 a9 21 24 a5 f8 c1 90 89 8e b8 e1 18 28 aa
e8 da 8c c4 bd 0a 5c f8 36 bd 5c da 33 3b 72 d9
52 f7 ba 62 94 9b 2c 9a 27 34 b5 c9 6b 0a 69 9a
44 3f bf a7 a4 a2 cb 4b ab 95 2f e8 b8 94 19 e9
6f e1 d9 00 cd 60 e7 2b 38 db a6 86 4e f0 83 fe
13 1a c2 d2 33 2f 8d 30 fa bb bd 0e c6 26 e4 55
e4 37 a4 78 62 33 4d f4 07 53 f2 a6 26 8b 15 94
55 ca a5 da 0d 4a eb f6 6d e5 bf 10 b9 28 17 83
49 1f 80 3c 50 56 68 0a 40 08 86 b1 8e ec 48 01
38 33 9c c7 4c 68 aa c9 a3 8d 28 57 8e eb 62 39
c0

DECRYPT TEXT: 256 Byte

45 8b 61 c3 84 ab 89 0b 71 ef ef b9 49 be a4 5b
b1 2b 71 e2 d5 55 3b e5 5a b0 f5 97 a9 dc 71 ed
66 d1 b0 ea 7c 38 f4 ec 26 e2 a5 6f 9f 48 ca 4f
73 3a 31 47 8f 6b 2c e9 1b 21 7f c3 fd f0 b0 63
c0 5f 4c 3c 96 3f 28 bc 21 cc 2b bf 14 f4 0e 86
2e 3e cd bc a9 f8 a4 c3 18 23 86 15 12 35 77 d2
93 c2 0e 29 00 35 e4 21 00 0e df 13 02 ed 99 2f
2a 65 ea d2 5c 8e 95 74 b0 1a 88 c2 4e ff 94 e1
c0 a2 0a c0 d6 ed e0 d5 fb bf e8 fc ab 80 2a d5
e4 14 a7 40 a2 3b b4 52 55 3c 13 a3 3a a7 83 f9
48 8c b9 1d 79 98 f2 74 57 da 70 01 59 9a d6 3c
ad 7c 7c 4f b7 2f a0 0b 6a b3 ad a4 59 30 9c a1
bc 55 be 34 ec b0 a8 42 89 17 43 e1 b0 18 1d 5d
94 98 ab 4a c7 4a 55 31 fc 01 d4 55 31 70 f6 ec
c4 b3 20 b0 63 c7 f2 eb dd 35 cc 8d 4d e8 e9 e0
80 94 2a 47 de 7f 77 da 7f 4b 2f b0 bb 24 9b 7f

E.1.1.1.3 Test 3

PLAIN TEXT: 1500 Bytes

d7 ba 2e 39 80 20 24 5d 54 ef e9 a0 d7 d2 7f 56
65 a9 9c 43 27 13 1c a6 5e 4a 55 18 6e f0 96 44
a9 c4 7d 29 e3 a1 85 36 8f 6e d5 65 3f 54 bb a4
fd 57 e6 23 6a 02 c9 c7 4c 1e de b9 0d 73 fd b6

36 7a de 19 1a 63 4e a9 d0 22 0e 0e 76 c8 b2 72
1f 97 95 88 99 5d 4e e4 7b 2c 9d 87 9f 99 3c d5
12 1a ed 2c 7c 3a d4 4b 5c e1 59 d1 a9 0a 42 c8
a1 d7 4f 39 33 9d 1d ad c9 b9 34 67 51 70 3c 63
89 28 8f 04 62 62 4f bd 43 a7 8e ec b0 d0 b3 50
a6 02 89 d9 9f a5 85 67 5d b9 ce ae 28 09 11 b0
31 9f b4 92 01 02 4f 43 a8 dc 2f 58 ab e2 a8 51
e3 30 29 81 d5 ad e8 31 65 b5 df 8d be ef 3c ee
8e ef 7f 8e f1 cd d1 99 a9 ff f0 54 e0 97 a4 c3
c7 cc 44 9b 79 2b cc de e0 ab 6a 9d 99 a6 8a 26
95 09 b4 85 d6 84 1d 7e 83 0d d1 63 a4 74 25 6a
40 69 05 b8 93 d1 96 73 7b ff 10 14 a5 99 39 39
a2 ed bd 77 71 da f4 f3 e7 c5 56 8a 39 7b f4 78
e3 f8 30 76 c8 c5 e8 42 c3 f7 55 68 90 8e a0 31
7b 5d a8 eb 36 9c de 1d 60 33 a6 98 ae 99 10 90
91 3f 05 59 03 ed 9a c6 e4 ef 2d 73 7d cc a4 f8
28 4b e2 5e e7 c0 7a 46 f3 20 de a0 b8 ed 30 49
2b 34 a1 2e 21 3b f3 04 2a 1f 77 a7 eb 1a 9e 13
65 80 70 4c 3f ea 91 31 09 6f d1 c1 5c 00 0a 87
34 aa b4 54 e4 a6 58 0d c5 ce b3 af e8 51 c1 4d
d0 31 98 0e 1a 29 3f 23 97 0f e4 f3 0f ed 79 42
97 2c 96 7a d1 ee 87 96 bb 3a 44 a3 8a 05 ef 59
35 86 67 4f af a6 72 45 b5 56 37 c3 43 af 05 d9
db 9a 53 ab 87 da 41 42 13 84 e4 9d 88 d3 f6 bd
59 5d 0c 07 02 7d 4b b6 d2 82 78 15 31 7c ed 0c
16 3f b7 9d 18 f7 df 2b 7a c2 c8 02 95 bd bf ed
19 ca f3 1a 47 3e d0 19 c0 47 2d f1 c3 19 fc d9
58 b2 75 70 a8 53 9a 22 15 61 24 a9 1e e2 96 36
ac 88 50 f2 c5 20 0a 84 67 37 74 2a 4f 70 02 a7
21 77 16 c8 ca b0 ea df 11 0d 87 2e ee 1d 64 99
a4 b4 8b 69 d3 94 ec 39 cb 60 62 19 cf 64 c0 f0
da d5 b7 a3 85 a0 81 95 ac 08 c2 9a 24 25 33 c8
d9 bd 30 ab 51 1c e4 1b 7b 46 34 4a a9 f3 39 82
c8 f0 25 4c 90 a5 e0 3c ad a2 d6 d1 c6 08 98 9f
c4 c7 49 14 e2 2d 2e 5d 72 61 a6 1a 54 df 9c 1b
cf c0 67 5e 65 46 9a 12 e7 6f e2 ad 76 79 4b 3a
3f 94 4e 21 c0 7b 7d 32 dc 23 4c 30 01 e7 4a d0
a7 b1 2d 0c f6 c7 1d dd 36 ff 8a ab 78 d5 e5 b7
68 32 d7 28 ad 53 59 89 76 a4 b8 76 8b 02 45 32
b2 72 3d a8 39 5a 84 6e 58 0d 19 d0 e2 fd 86 49
2f 5c 71 db af ca 63 24 6e 1b 9a f8 1c df 29 ce
51 66 75 89 bf f9 f6 17 06 0e e6 e7 0b 6c 30 39
c8 a0 13 77 69 76 9b d6 91 34 ce ad 13 f7 7a 63
5c ef eb 1b e7 e1 32 ec ee 17 d3 f8 83 02 31 4a
a1 44 c0 0a b9 5a e0 49 8e ad f6 a0 a4 6f 03 ff

5e ed 1a 44 ce 4b 30 bb 62 02 b3 e4 03 e3 2e a4
26 ed ad df 47 8d 28 d5 3a 1d 74 dd 8c 77 dc e9
63 f5 2d 31 40 5d eb a1 5e 9e 85 61 81 b2 05 a7
9f b2 86 e6 3e ad ba 77 ca 2e 54 56 a4 2f 3f 07
24 6b 37 63 c8 22 04 26 bf 88 87 40 3a 8b e6 d9
3d 6b be 7b 18 77 f1 e2 a4 45 37 48 73 76 4e 97
e1 84 f9 a8 a5 fd cd 64 84 53 a3 be de 89 96 1a
f4 53 94 0c ca 85 ed 6e c9 24 b5 3c 99 03 d2 7a
86 cb 21 2b c7 ed 8f 4b 40 32 09 1d bb 9e 37 ae
f1 ca b9 bb 4f a6 28 18 c9 dd 53 62 df 25 db 64
ef fc 8f b6 e9 1e 01 28 4f 09 45 09 a6 7b b7 97
45 70 51 93 15 78 aa de 54 fd 40 32 21 1a 96 10
16 25 c5 fe 42 c5 25 91 cd 6a 9a 73 e4 50 0a 29
c0 5a bc d4 d2 65 b2 26 62 f1 58 82 0b ed 92 20
12 57 1d 53 1c 42 e4 e9 ac 7d 5b 90 cd 65 b8 8d
be 73 60 8f d8 12 b5 39 02 0c bb 0c f9 4c 2c 0a
a3 49 5d be 8a 40 a6 35 bd 01 c4 8a 65 7c 16 23
ee 76 b2 c5 87 66 fe 89 71 b8 95 69 04 c0 72 a6
08 cf 64 92 0f 09 c7 cb 0a 8b 55 6e 06 6a 91 f3
e0 42 b8 67 a7 b5 ef 17 6d 84 80 71 44 f2 17 4b
c0 7a dd ce 83 a3 99 8c 2d ee fa 33 58 8a 25 37
cb dd 9d 72 92 8c 89 ff 10 08 6f 53 fa 85 9d b9
ff 7a 87 81 1c 20 0c 49 0d 06 7b 64 8f a0 9b 5a
7d 38 cc 0e c4 54 0d d3 5c 7b 25 55 00 c2 0e ff
3b 95 7f 57 b4 8b a0 c1 90 1b 25 1f ba c0 79 37
f7 44 45 ba 98 51 8d f3 cc b1 47 cc 73 54 ca ae
e9 48 05 9c d2 a4 5d 62 be 82 81 78 41 f9 ae 38
3d f2 f1 d4 43 7e c6 0e 2e 0d d9 a1 61 a2 4e 49
e9 52 e5 bb f5 42 1c b3 c3 9c 2b 04 95 d9 3b d1
ca 2b a5 0c a8 6a 1a d6 77 f2 76 d7 93 c4 20 7c
15 04 37 0a 45 53 bd 08 ef e7 0b 83 bf 45 54 89
70 f8 95 18 62 ae ee d9 a0 64 b0 33 27 cf af 3c
d3 e5 45 18 37 01 1f 26 e8 29 a9 a6 6e fc 2f dd
f4 c3 f5 56 71 e2 2e 10 45 dd 42 6b ac f0 a6 7e
d5 eb 95 0c ec b4 31 d3 dd da 79 4a d6 a7 27 c9
69 1b 1f da fd 4c e9 41 29 2b ac d4 1a 52 52 ef
3d e6 fa 28 99 2b fb 75 04 73 bf d9 19 e5 a2 82
00 c0 5c fc 0c 44 3d 35 6e e8 08 88 3a 59 76 76
3f 70 9d d8 9b 97 4c 9e 09 0a 77 22 ef 18 a4 ee
d8 ff e9 e3 43 25 17 b1 0d 1f 38 46 78 ae bb b7
1e 57 8e b8 ee d9 56 f7 e3 cc 19 d1 e4 bd bf bb
bc a8 9e fe cc b5 ae d9 d3 e6 1e 4b 93 d9 01 b0
30 8e 68 1d 67 bd 14 49 88 2c 1a 6b e8 d8 25 a4
7f c3 a1 4b 77 4f 24 4a 34 42 94 c6 1a 95 76 4a
23 de 67 89 9a 7a d2 22 a6 ec 8c 8e

Roll-over-counter: 1 Byte

00

PHY Syncronization: 3 Byte

ffffff

Counter: 16 Bytes

00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff

Key: 16Bytes

00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff

CIPHER TEXT: 1500 Byte + 1 Byte(Roll-over-counter)

00 6a 3e f1 80 dc 3d 4a 24 b1 e9 26 d9 b9 28 cf
96 0d 4c df 31 7e 30 ba a2 4a e2 56 df fe 01 01
27 11 98 2d 7f dd 45 ca 7a 68 31 7d 82 44 db 8a
6c 34 8b 19 c4 a3 b4 9b 55 e8 59 cb c5 d9 2c 01
79 1a 5e 58 a9 1d 1d 27 e0 e9 76 9b b5 8e bf c7
47 2f a1 3d a7 e9 d1 11 e5 3b cb ca 7b 9a 56 e3
0f 88 71 c2 21 d9 f7 f1 fa d5 61 3e 23 b3 cf 71
0f 51 3e 61 56 65 4f 70 ef c4 ff 66 96 24 fd 71
d0 be 30 e7 50 2f a3 35 4f 8c ad af 7b 11 39 03
c1 7d a9 89 3d 9f 55 7a 9e 9d aa 35 b5 86 b7 7b
26 98 ca 0d 42 18 7d 96 0f 24 a0 d9 17 02 fb 80
7e 54 8e 87 fd 4d 0f 78 c0 b4 bb 7c ef c1 3b f5
ab 05 1e b9 d8 2e 30 8d dc 73 1a 15 93 db 9a 2d
cb 99 f1 35 dc f4 8a 6f 82 f9 15 ae 71 80 c1 ff
83 4e 3a a8 65 e3 2b e5 d5 56 be ac 60 05 d4 cd
b2 f3 61 e8 b3 25 04 28 0a 89 9c 68 2a f5 df 9b
86 fb 5d aa 90 d1 af 43 06 c0 9f d4 9d 1c 0d cb
c9 d2 ef 3e bc 9c 92 a4 61 36 b8 f6 4b d7 6d 8f
2e d3 ec 87 40 9d 19 a0 e4 10 f7 74 76 7d f1 60
4b 98 70 ce 5d aa 28 0e 42 dc 4e a3 86 0d 30 5f
f6 2f 6a b8 a0 c3 f7 85 56 58 53 d4 e0 16 a8 be
09 0a d9 d9 1f ab 02 b9 94 74 75 1c f4 ca 6d 75
93 4f 08 1c b6 46 46 ff 97 c2 c0 af 64 39 a3 4c
e1 90 74 6d d0 a8 2b 75 0f 21 a5 3b ad 95 6f 42
c2 9d 23 19 e4 f3 3f 45 c2 eb c5 bd 2c 75 5f 77

d9 01 0f ef e9 01 53 2e 34 17 21 ba 5a d8 bc 60
1a f2 32 e1 f2 88 fc a2 ac 55 7d 47 b2 1b cf cd
79 c1 9a ac 24 19 e0 5e 56 f8 db c3 3b c9 aa f4
fd 19 80 83 2a 59 4a c6 5e f8 40 13 00 d9 77 5c
4a ca b0 5b 53 de 4c a0 3d 96 04 60 d8 bf d5 a3
f9 da 41 ed 8e 0b e9 a8 52 40 46 c1 21 78 4e 04
5b 0b 20 ad 4b 23 32 51 8e e7 b2 ac a1 51 c8 4d
0f e0 cd e5 64 16 71 df 64 76 1d 51 71 02 a2 76
66 38 01 46 87 ec 44 6f 1d 44 f5 8b 7b 7e 63 14
4b 29 2b 8d fc a6 3e 14 75 b8 a8 27 23 b3 f2 1c
79 a8 2f 2b 3b 69 3e d9 0d 74 1a 32 c6 b0 98 86
18 3f 9f a2 c0 a5 bb b8 85 61 7b 0d 23 47 e4 39
0f 53 cf 9c 19 34 55 3c de 0e 24 ed 07 6c e9 d3
7d 91 aa 32 33 5a df 42 90 55 63 eb cf f7 b8 62
9b 40 2a 3c e8 06 ef 84 50 da 51 fc 82 26 63 fb
c2 58 44 d6 1b 56 db 52 d9 7f c0 36 63 8f 70 2a
6e be ee 72 23 40 3e db 47 96 d3 eb 4c 9e dc cf
db f5 70 dd 01 ec a6 93 95 f0 91 06 c4 6f 74 5e
c7 cd 74 e7 45 53 af 92 c3 96 f3 4d 10 85 b2 11
4f cc 19 b0 a6 11 f5 ac 5a 2b 2b 96 44 3d 35 b4
3a ad 66 32 d6 b5 66 4a 55 2f aa df a2 2d d9 0e
e3 90 90 dd 0b 09 5a 3a 27 24 98 ff 27 2b 45 0e
f4 22 42 1c f2 4a 2d 00 98 b5 ea 88 87 a9 fe 8d
58 8a 4e 64 85 82 13 d1 13 43 66 b4 91 85 20 8d
b7 e2 a5 20 17 02 78 f7 19 a0 b4 74 a2 80 ec 5f
8d 47 7c 23 7f df 35 df d4 ad c8 ac 9a e5 df 6e
e0 5c 81 ce 6a 6c 75 b1 07 09 b7 f0 97 73 82 56
93 e6 fe 5d b1 8e 85 08 f3 df 53 50 4d 3c 12 71
45 50 45 47 c1 64 f0 09 38 26 3d 56 e6 ed de 71
17 28 3b 51 08 c0 e9 d1 13 ae 7c ff 82 73 0b ee
ae 8d 3d a1 e2 47 a4 0c 66 67 83 61 6e 1a 49 7f
d6 9d d0 21 70 fc 99 f5 f0 0c 0f 1a 5e b6 94 ac
4a 7d f6 f1 89 dd 49 2d f9 0f 1c 21 ae 8c 5c c3
0c 38 3e 24 9d 32 9f 4a 0c a6 83 93 f1 e5 1c 4e
c3 a2 5d 77 95 51 60 7b e5 09 eb 4c b7 5b 1b 7f
09 89 e0 63 5c 36 1f 42 9b 18 66 39 28 18 ad ec
c7 56 d8 95 4f e8 b8 02 8f 0a 57 6b 7e d1 1c 61
e2 b8 40 54 12 d0 01 be f6 e4 e4 01 09 86 56 48
12 a5 85 62 d9 34 00 8e c5 b3 77 e2 e7 08 5e 9d
61 d8 dc b1 4c 3c 21 75 31 5d 75 ce 5d 15 27 fe
7f 85 21 10 d3 41 07 74 0b 35 af 1a 45 9c db 4e
a1 f4 f4 c0 28 d9 d4 54 ad ef e2 d6 1e 94 10 f8
3e ed ae ef 6d e9 d1 19 b5 2b ed d2 54 b8 b0 47
5b 8b c7 e6 2c 82 ed 7d d6 f4 f8 59 60 7c 15 12
fd 68 41 fe 46 77 f8 96 99 45 dd ed 47 68 4d 6d

e8 51 cc da 66 a0 56 4d e9 74 af f8 06 ff 92 7e
6d 12 ba 21 3c 86 04 cf a0 c0 bd 18 86 c2 11 bc
13 81 a2 54 60 a8 21 fd 50 b2 19 5c 8c 5a ee 00
fe 11 b8 71 ea 15 c3 15 28 1c 41 d5 a7 27 22 c2
42 e7 5f 1b aa ec f9 09 04 00 0d 0d 63 8b 84 aa
a0 d2 e7 f5 1b b6 d8 5f f7 5c 53 f0 9a 41 f2 27
96 33 c4 93 d9 11 5c 5b 1a a4 d4 f8 2c c0 fc 79
99 ad 8b cd 34 fb 7e e6 60 40 11 80 30 b2 0d 23
36 df d8 b5 0c d2 76 1a 1f 4d 7b d9 32 3e 97 09
f7 5f b1 6c 3c 6b 78 17 c0 4e 63 66 a7 8b 46 85
38 bd fa d1 e2 e9 3d c8 33 33 94 08 b2 c2 8b c8
ab 89 1f 78 d8 7c f7 0b 61 f2 f2 6c 81 38 72 f5
9d d3 32 43 7b 15 68 e3 d8 eb be 73 d1 1d 35 16
a1 17 dc 02 65 da 91 62 2a 9f 82 6d 75 f7 ab 0c
83 63 e2 7f d9 25 9b 44 9e 35 fd 0e 1a 1e b1 c7
e4 46 a6 03 2a 11 ba d1 2a aa 34 6b ee d1 ae 3b
c4 bc cb f9 35 03 e0 e6 03 55 1f bf b0 c0 b4 7d
99 ad 7d 5b 65 63 a7 9c a4 61 8b 5d 11 bf 40 43
bb 83 4d d8 fa ec 25 60 e2 a2 3c b0 6e 23 92 4b
ff 47 83 7f 06 4d 27 67 d8 50 80 07 69 ae e3 d0
7b 9e 18 7a 1f 46 52 b5 4e 6a bc 34 f7 91 60 ee
5b f9 2c a7 ce 8f 90 d0 e5 6f d1 44 f0 2f 98 d3
26 79 80 7a 7c 76 bf 86 25 e6 d1 c6 0a 24 7b 61
63 ff 6a f3 f5 d5 8b ce 4f c5 2c d6 0c

DECRYPT TEXT: 1500 Byte

d7 ba 2e 39 80 20 24 5d 54 ef e9 a0 d7 d2 7f 56
65 a9 9c 43 27 13 1c a6 5e 4a 55 18 6e f0 96 44
a9 c4 7d 29 e3 a1 85 36 8f 6e d5 65 3f 54 bb a4
fd 57 e6 23 6a 02 c9 c7 4c 1e de b9 0d 73 fd b6
36 7a de 19 1a 63 4e a9 d0 22 0e 0e 76 c8 b2 72
1f 97 95 88 99 5d 4e e4 7b 2c 9d 87 9f 99 3c d5
12 1a ed 2c 7c 3a d4 4b 5c e1 59 d1 a9 0a 42 c8
a1 d7 4f 39 33 9d 1d ad c9 b9 34 67 51 70 3c 63
89 28 8f 04 62 62 4f bd 43 a7 8e ec b0 d0 b3 50
a6 02 89 d9 9f a5 85 67 5d b9 ce ae 28 09 11 b0
31 9f b4 92 01 02 4f 43 a8 dc 2f 58 ab e2 a8 51
e3 30 29 81 d5 ad e8 31 65 b5 df 8d be ef 3c ee
8e ef 7f 8e f1 cd d1 99 a9 ff f0 54 e0 97 a4 c3
c7 cc 44 9b 79 2b cc de e0 ab 6a 9d 99 a6 8a 26
95 09 b4 85 d6 84 1d 7e 83 0d d1 63 a4 74 25 6a
40 69 05 b8 93 d1 96 73 7b ff 10 14 a5 99 39 39
a2 ed bd 77 71 da f4 f3 e7 c5 56 8a 39 7b f4 78

e3 f8 30 76 c8 c5 e8 42 c3 f7 55 68 90 8e a0 31
7b 5d a8 eb 36 9c de 1d 60 33 a6 98 ae 99 10 90
91 3f 05 59 03 ed 9a c6 e4 ef 2d 73 7d cc a4 f8
28 4b e2 5e e7 c0 7a 46 f3 20 de a0 b8 ed 30 49
2b 34 a1 2e 21 3b f3 04 2a 1f 77 a7 eb 1a 9e 13
65 80 70 4c 3f ea 91 31 09 6f d1 c1 5c 00 0a 87
34 aa b4 54 e4 a6 58 0d c5 ce b3 af e8 51 c1 4d
d0 31 98 0e 1a 29 3f 23 97 0f e4 f3 0f ed 79 42
97 2c 96 7a d1 ee 87 96 bb 3a 44 a3 8a 05 ef 59
35 86 67 4f af a6 72 45 b5 56 37 c3 43 af 05 d9
db 9a 53 ab 87 da 41 42 13 84 e4 9d 88 d3 f6 bd
59 5d 0c 07 02 7d 4b b6 d2 82 78 15 31 7c ed 0c
16 3f b7 9d 18 f7 df 2b 7a c2 c8 02 95 bd bf ed
19 ca f3 1a 47 3e d0 19 c0 47 2d f1 c3 19 fc d9
58 b2 75 70 a8 53 9a 22 15 61 24 a9 1e e2 96 36
ac 88 50 f2 c5 20 0a 84 67 37 74 2a 4f 70 02 a7
21 77 16 c8 ca b0 ea df 11 0d 87 2e ee 1d 64 99
a4 b4 8b 69 d3 94 ec 39 cb 60 62 19 cf 64 c0 f0
da d5 b7 a3 85 a0 81 95 ac 08 c2 9a 24 25 33 c8
d9 bd 30 ab 51 1c e4 1b 7b 46 34 4a a9 f3 39 82
c8 f0 25 4c 90 a5 e0 3c ad a2 d6 d1 c6 08 98 9f
c4 c7 49 14 e2 2d 2e 5d 72 61 a6 1a 54 df 9c 1b
cf c0 67 5e 65 46 9a 12 e7 6f e2 ad 76 79 4b 3a
3f 94 4e 21 c0 7b 7d 32 dc 23 4c 30 01 e7 4a d0
a7 b1 2d 0c f6 c7 1d dd 36 ff 8a ab 78 d5 e5 b7
68 32 d7 28 ad 53 59 89 76 a4 b8 76 8b 02 45 32
b2 72 3d a8 39 5a 84 6e 58 0d 19 d0 e2 fd 86 49
2f 5c 71 db af ca 63 24 6e 1b 9a f8 1c df 29 ce
51 66 75 89 bf f9 f6 17 06 0e e6 e7 0b 6c 30 39
c8 a0 13 77 69 76 9b d6 91 34 ce ad 13 f7 7a 63
5c ef eb 1b e7 e1 32 ec ee 17 d3 f8 83 02 31 4a
a1 44 c0 0a b9 5a e0 49 8e ad f6 a0 a4 6f 03 ff
5e ed 1a 44 ce 4b 30 bb 62 02 b3 e4 03 e3 2e a4
26 ed ad df 47 8d 28 d5 3a 1d 74 dd 8c 77 dc e9
63 f5 2d 31 40 5d eb a1 5e 9e 85 61 81 b2 05 a7
9f b2 86 e6 3e ad ba 77 ca 2e 54 56 a4 2f 3f 07
24 6b 37 63 c8 22 04 26 bf 88 87 40 3a 8b e6 d9
3d 6b be 7b 18 77 f1 e2 a4 45 37 48 73 76 4e 97
e1 84 f9 a8 a5 fd cd 64 84 53 a3 be de 89 96 1a
f4 53 94 0c ca 85 ed 6e c9 24 b5 3c 99 03 d2 7a
86 cb 21 2b c7 ed 8f 4b 40 32 09 1d bb 9e 37 ae
f1 ca b9 bb 4f a6 28 18 c9 dd 53 62 df 25 db 64
ef fc 8f b6 e9 1e 01 28 4f 09 45 09 a6 7b b7 97
45 70 51 93 15 78 aa de 54 fd 40 32 21 1a 96 10
16 25 c5 fe 42 c5 25 91 cd 6a 9a 73 e4 50 0a 29

c0 5a bc d4 d2 65 b2 26 62 f1 58 82 0b ed 92 20
12 57 1d 53 1c 42 e4 e9 ac 7d 5b 90 cd 65 b8 8d
be 73 60 8f d8 12 b5 39 02 0c bb 0c f9 4c 2c 0a
a3 49 5d be 8a 40 a6 35 bd 01 c4 8a 65 7c 16 23
ee 76 b2 c5 87 66 fe 89 71 b8 95 69 04 c0 72 a6
08 cf 64 92 0f 09 c7 cb 0a 8b 55 6e 06 6a 91 f3
e0 42 b8 67 a7 b5 ef 17 6d 84 80 71 44 f2 17 4b
c0 7a dd ce 83 a3 99 8c 2d ee fa 33 58 8a 25 37
cb dd 9d 72 92 8c 89 ff 10 08 6f 53 fa 85 9d b9
ff 7a 87 81 1c 20 0c 49 0d 06 7b 64 8f a0 9b 5a
7d 38 cc 0e c4 54 0d d3 5c 7b 25 55 00 c2 0e ff
3b 95 7f 57 b4 8b a0 c1 90 1b 25 1f ba c0 79 37
f7 44 45 ba 98 51 8d f3 cc b1 47 cc 73 54 ca ae
e9 48 05 9c d2 a4 5d 62 be 82 81 78 41 f9 ae 38
3d f2 f1 d4 43 7e c6 0e 2e 0d d9 a1 61 a2 4e 49
e9 52 e5 bb f5 42 1c b3 c3 9c 2b 04 95 d9 3b d1
ca 2b a5 0c a8 6a 1a d6 77 f2 76 d7 93 c4 20 7c
15 04 37 0a 45 53 bd 08 ef e7 0b 83 bf 45 54 89
70 f8 95 18 62 ae ee d9 a0 64 b0 33 27 cf af 3c
d3 e5 45 18 37 01 1f 26 e8 29 a9 a6 6e fc 2f dd
f4 c3 f5 56 71 e2 2e 10 45 dd 42 6b ac f0 a6 7e
d5 eb 95 0c ec b4 31 d3 dd da 79 4a d6 a7 27 c9
69 1b 1f da fd 4c e9 41 29 2b ac d4 1a 52 52 ef
3d e6 fa 28 99 2b fb 75 04 73 bf d9 19 e5 a2 82
00 c0 5c fc 0c 44 3d 35 6e e8 08 88 3a 59 76 76
3f 70 9d d8 9b 97 4c 9e 09 0a 77 22 ef 18 a4 ee
d8 ff e9 e3 43 25 17 b1 0d 1f 38 46 78 ae bb b7
1e 57 8e b8 ee d9 56 f7 e3 cc 19 d1 e4 bd bf bb
bc a8 9e fe cc b5 ae d9 d3 e6 1e 4b 93 d9 01 b0
30 8e 68 1d 67 bd 14 49 88 2c 1a 6b e8 d8 25 a4
7f c3 a1 4b 77 4f 24 4a 34 42 94 c6 1a 95 76 4a
23 de 67 89 9a 7a d2 22 a6 ec 8c 8e

E.1.1.1.4 Test Program

```
/*****************************************************************/
/* 802.16e MBS (Multimedia Broadcast Service) AES-CTR mode example */
/* program for KAT (Known Answer Test). KAT help implementors to   */
/* verify AES algorithm and CTR mode correctly for MBS defined     */
/* in PKMv2                                                        */
/* Version Number: 0.2                                             */
/* Name: JunHyuk Song, Jicheol Lee                                 */
/*****************************************************************/

#include <stdlib.h>
#include <stdio.h>

#define MAX_BUF              10000

/*****************************/
/*** AES 16X16 SBOX Table ****/
/*****************************/

unsigned char sbox_table[256] =
{
   0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
   0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
   0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
   0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
   0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
   0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
   0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
   0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
   0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
   0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
   0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
   0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
   0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
   0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
   0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
   0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};

/*****************************/
/**** Function Prototypes ****/
/*****************************/

void bitwise_xor(unsigned char *ina, unsigned char *inb, unsigned char *out);
void print_hex(unsigned char *buf, int len) ;

/*****************************************************************/
/*************** AES algorithm operation functions ***************/
/*****************************************************************/

void xor_128(unsigned char *a, unsigned char *b, unsigned char *out);
void xor_32(unsigned char *a, unsigned char *b, unsigned char *out);
```

```
unsigned char sbox(unsigned char a);
void next_key(unsigned char *key, int round);
void byte_sub(unsigned char *in, unsigned char *out);
void shift_row(unsigned char *in, unsigned char *out);
void mix_column(unsigned char *in, unsigned char *out);
void add_round_key( unsigned char *shiftrow_in,
                unsigned char *mcol_in,
                unsigned char *block_in,
                int round,
                unsigned char *out);
void aes128k128d(unsigned char *key, unsigned char *data, unsigned char *ciphertext);

/*****************************************/
/* This function is to generate 32bit nonce */
/* based on GCC rand()                   */
/*****************************************/

unsigned long random_32bit(void)
{
        return (unsigned long) rand();
}
/************************************************/
/* This function is to generate random plain text */
/************************************************/

unsigned char random_8bit(void)
{
        unsigned char ret;

        ret = (unsigned char) 1 + (int) (256.0*rand()/(RAND_MAX+1.0));
        return ret;
}

void generate_plain(unsigned char *plain, int len)
{
        int          i;

        for ( i=0; i<len; i++ ) {
                plain[i] = random_8bit();
        }
}




/*************************************************************************/
/* AES Encryption functions are defined here.                            */
/* Performs a 128 bit AES encryption with 128 bit key and data blocks based */
/* based on NIST Special Publication 800-38A, FIPS 197                   */
/*************************************************************************/

/************************/
```

```
/* 128 bits XOR function */
/*************************/


void xor_128(unsigned char *a, unsigned char *b, unsigned char *out)
{
   int i;
   for (i=0;i<16; i++)
   {
      out[i] = a[i] ^ b[i];
   }
}


/*************************/
/* 32 bits XOR function */
/*************************/

void xor_32(unsigned char *a, unsigned char *b, unsigned char *out)
{
   int i;
   for (i=0;i<4; i++)
   {
      out[i] = a[i] ^ b[i];
   }
}


/***********************************/
/* AES SBOX Table Setup   **********/
/***********************************/

unsigned char sbox(unsigned char a)
{
   return sbox_table[(int)a];
}


/*****************************************/
/* AES next_key operation ****************/
/*****************************************/

void next_key(unsigned char *key, int round)
{
   unsigned char rcon;
   unsigned char sbox_key[4];
   unsigned char rcon_table[12] =
   {
      0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
      0x1b, 0x36, 0x36, 0x36
   };
```

```
    sbox_key[0] = sbox(key[13]);
    sbox_key[1] = sbox(key[14]);
    sbox_key[2] = sbox(key[15]);
    sbox_key[3] = sbox(key[12]);

    rcon = rcon_table[round];

    xor_32(&key[0], sbox_key, &key[0]);
    key[0] = key[0] ^ rcon;

    xor_32(&key[4], &key[0], &key[4]);
    xor_32(&key[8], &key[4], &key[8]);
    xor_32(&key[12], &key[8], &key[12]);
}


/*************************************/
/* AES Byte Substituition **********/
/*************************************/

void byte_sub(unsigned char *in, unsigned char *out)
{
    int i;
    for (i=0; i< 16; i++)
    {
        out[i] = sbox(in[i]);
    }
}



/*************************************/
/* AES Shift Row Operation *********/
/*************************************/

void shift_row(unsigned char *in, unsigned char *out)
{
    out[0] =  in[0];
    out[1] =  in[5];
    out[2] =  in[10];
    out[3] =  in[15];
    out[4] =  in[4];
    out[5] =  in[9];
    out[6] =  in[14];
    out[7] =  in[3];
    out[8] =  in[8];
    out[9] =  in[13];
    out[10] = in[2];
    out[11] = in[7];
    out[12] = in[12];
    out[13] = in[1];
    out[14] = in[6];
```

```
   out[15] = in[11];
}


/***********************************/
/****** AES mix_column operation ***/
/***********************************/

void mix_column(unsigned char *in, unsigned char *out)
{
   int i;
   unsigned char add1b[4];
   unsigned char add1bf7[4];
   unsigned char rotl[4];
   unsigned char swap_halfs[4];
   unsigned char andf7[4];
   unsigned char rotr[4];
   unsigned char temp[4];
   unsigned char tempb[4];

   for (i=0 ; i<4; i++)
   {
      if ((in[i] & 0x80)== 0x80)
         add1b[i] = 0x1b;
      else
         add1b[i] = 0x00;
   }

   swap_halfs[0] = in[2];    /* Swap halfs */
   swap_halfs[1] = in[3];
   swap_halfs[2] = in[0];
   swap_halfs[3] = in[1];

   rotl[0] = in[3];        /* Rotate left 8 bits */
   rotl[1] = in[0];
   rotl[2] = in[1];
   rotl[3] = in[2];

   andf7[0] = in[0] & 0x7f;
   andf7[1] = in[1] & 0x7f;
   andf7[2] = in[2] & 0x7f;
   andf7[3] = in[3] & 0x7f;

   for (i = 3; i>0; i--)    /* logical shift left 1 bit */
   {
      andf7[i] = andf7[i] << 1;
      if ((andf7[i-1] & 0x80) == 0x80)
      {
         andf7[i] = (andf7[i] | 0x01);
      }
   }
   andf7[0] = andf7[0] << 1;
```

```
    andf7[0] = andf7[0] & 0xfe;

    xor_32(add1b, andf7, add1bf7);

    xor_32(in, add1bf7, rotr);

    temp[0] = rotr[0];          /* Rotate right 8 bits */
    rotr[0] = rotr[1];
    rotr[1] = rotr[2];
    rotr[2] = rotr[3];
    rotr[3] = temp[0];

    xor_32(add1bf7, rotr, temp);
    xor_32(swap_halfs, rotl,tempb);
    xor_32(temp, tempb, out);
}

/* AES Encryption function that will do multiple round of AddRoundKey, SubBytes,
   ShiftRows, and MixColumns operations */

void aes128k128d(unsigned char *key, unsigned char *data, unsigned char *ciphertext)
{
    int round;
    int i;
    unsigned char intermediatea[16];
    unsigned char intermediateb[16];
    unsigned char round_key[16];

    for(i=0; i<16; i++) round_key[i] = key[i];

    for (round = 0; round < 11; round++)
    {
        if (round == 0) /* First AddRound Key Operation */
        {
            xor_128(round_key, data, ciphertext);
            next_key(round_key, round);
        }
        else if (round == 10) /* Final Round operations */
        {
            byte_sub(ciphertext, intermediatea);
            shift_row(intermediatea, intermediateb);
            xor_128(intermediateb, round_key, ciphertext);
        }
        else    /* 1 - 9 */
        {
            byte_sub(ciphertext, intermediatea);
            shift_row(intermediatea, intermediateb);
            mix_column(&intermediateb[0], &intermediatea[0]);
            mix_column(&intermediateb[4], &intermediatea[4]);
            mix_column(&intermediateb[8], &intermediatea[8]);
            mix_column(&intermediateb[12], &intermediatea[12]);
            xor_128(intermediatea, round_key, ciphertext);
```

```
            next_key(round_key, round);
        }
    }

}

/************************************/
/* bitwise_xor()                    */
/* A 128 bit, bitwise exclusive or  */
/************************************/

void bitwise_xor(unsigned char *ina, unsigned char *inb, unsigned char *out)
{
    int i;
    for (i=0; i<16; i++)
    {
        out[i] = ina[i] ^ inb[i];
    }
}


/*************************************************/
/* It generate 128bit key as                     */
/* 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff */
/* for Variable Key Known Answer Test            */
/*************************************************/

void generate_key(unsigned char *key)
{
        int                                     i;

        for (i=0; i<8; i++ ) {
                key[i] = 0x00;
        }
        for (i=8; i<16; i++ ) {
                key[i] = 0xff;
        }
}

/*************************************************/
/* Initialization of Counter
/* first, construct 32 bit value by concatenate 8bit-rollovercounter
 *                      and 24bit-phy_sync
 * seconds, concatnate the above results 4 times
/*************************************************/

void init_counter(unsigned char rollcnt, unsigned long phy_sync, unsigned char *ctr)
{
        int                     i, j;

        for ( i=0; i<4; i++ ) {
                ctr[i*4+0] = rollcnt;
```

```
                ctr[i*4+1] = (phy_sync >> 16 ) & 0xff;
                ctr[i*4+2] = (phy_sync >> 8 ) & 0xff;
                ctr[i*4+3] = phy_sync & 0xff;
        }
}

/*************************************************************/
/* It increment counter by one upon encryption of each block */
/*************************************************************/

void add_counter(char *ctr)
{
        int             value, i;
        int             overflow;

        overflow = 1;
        for ( i=15; i>=0 ; i-- ) {
                if ( overflow == 0 ) break;
                value = ctr[i] & 0xff;
                value ++;
                if ( value >= 256 )
                        overflow = 1;
                else overflow = 0;
                ctr[i] = value & 0xff;
        }
}
/* Return Roll over Counter */
unsigned char get_rollcnt(void)
{
        return 0x00;
}
unsigned long get_phy_sync(void)
{
        /* Suppose that phy sync 24bits are all one in this example. */
        return 0x00ffffff;
}

/*******************************************************/
/* int encrypt_pdu()                                   */
/* Encrypts a plaintext pdu in accordance with         */
/* the proposed 802.16e AES CTR specification.         */
/* Roll-over-counter takes place.                      */
/* Returns the resulting cipher text                   */
/*******************************************************/
int encrypt_pdu(unsigned char *key, unsigned char *plain, int len, unsigned char *cipher)
{
        int                             i, n_blocks, n_remain, out_len = 0;
        unsigned char   ctr[16], rollcnt;
        unsigned char   aes_out[16], remain[16], temp[16];
        unsigned long   phy_sync_value;

        rollcnt = get_rollcnt();
```

```
		phy_sync_value = get_phy_sync();
#ifdef DEBUG
		printf("Roll-over-counter: 1 Byte\n\n");
		printf("%02x\n\n",rollcnt);
		printf("PHY Syncronization: 3 Byte\n\n");
		printf("%06x\n\n", phy_sync_value);
#endif

		cipher[0] = rollcnt;

		out_len += 1;

		n_blocks = len / 16;
		n_remain  = len % 16;

		init_counter(rollcnt,phy_sync_value,ctr);
#ifdef DEBUG
		printf("Counter: 16 Bytes\n\n");
		print_hex(ctr,16);
		printf("\n");
   printf("Key: 16Bytes\n\n");
   print_hex(key,16);
		printf("\n");
#endif
		for ( i=0; i< n_blocks; i++ ) {
			aes128k128d(key, ctr, aes_out);
		bitwise_xor(aes_out, &plain[i*16], &cipher[i*16+1]);
			add_counter(ctr);

			out_len += 16;
		}

		for ( i=0; i<16; i++ ) {
			remain[i] = 0;
		}
		for ( i=0; i<n_remain; i++ ) {
			remain[i] = plain[n_blocks*16+i];
		}
		aes128k128d(key,ctr,aes_out);
		bitwise_xor(aes_out,&remain[0], &temp[0]);

		for ( i=0; i<n_remain; i++ ) {
			cipher[n_blocks*16+1+i] = temp[i];
		}
		out_len += n_remain;
		return out_len;
}
/*****************************************************/
/* int decrypt_pdu()                                 */
/* decrypts a cipher pdu in accordance with          */
/* the proposed 802.16e AES CTR specification.       */
/* Decode roll-over-counter field                    */
```

```
/* Returns the resulting decrypted text                */
/*****************************************************/

int decrypt_pdu(unsigned char *key, unsigned char *cipher, int len, unsigned char *plain)
{
        int                                     i, n_blocks, n_remain, out_len = 0;
        unsigned char       ctr[16],rollcnt;
        unsigned char       aes_out[16], remain[16], temp[16];
        unsigned long       phy_sync_value;

        phy_sync_value = get_phy_sync();
        rollcnt = cipher[0];

        len -= 1;

        n_blocks = len / 16;
        n_remain  = len % 16;

        init_counter(rollcnt, phy_sync_value, ctr);
        for ( i=0; i< n_blocks; i++ ) {
            aes128k128d(key, ctr, aes_out);
            bitwise_xor(aes_out, &cipher[i*16+1], &plain[i*16]);
                    add_counter(ctr);
                    out_len += 16;
        }

        for ( i=0; i<16; i++ ) {
                remain[i] = 0;
        }
        for ( i=0; i<n_remain; i++ ) {
                remain[i] = cipher[n_blocks*16+1+i];
        }
        aes128k128d(key,ctr,aes_out);
        bitwise_xor(aes_out,&remain[0], &temp[0]);

        for ( i=0; i<n_remain; i++ ) {
                plain[n_blocks*16+i] = temp[i];
        }
        out_len += n_remain;
        return out_len;
}

/* HEX value print out function          */
void print_hex(unsigned char *buf, int len)
{
        int                     i;

        for ( i=0; i<len; i++ ) {
                printf("%02x ", buf[i]);
                if ( (i % 16) == 15 ) printf("\n");
        }
        if ( (i % 16) != 0 ) printf("\n");
```

```
}

int compare(unsigned char *x, unsigned char *y, int len)
{
	int			i;

	for ( i=0; i<len; i++ ) {
		if ( x[i] == y[i] ) continue;
		return (x[i] - y[i]);
	}
	return 0;
}


int test_case(int length)
{
	unsigned char	key[16];
	unsigned char	plain[MAX_BUF];
	unsigned char	cipher[MAX_BUF+4];
	unsigned char	decrypt[MAX_BUF];

	/* 0. Get a 128bits key */
	generate_key(key);

	/* 1. Generate Plain Text with length */

	generate_plain(plain,length);

#ifdef DEBUG
	printf("PLAIN TEXT: %d Bytes\n\n",length);
	print_hex(plain,length);
	printf("\n\n");
#endif

	/* 2. Encrypt Plain Text to Cipher Text */

	encrypt_pdu(key,plain,length,cipher);

#ifdef DEBUG
	printf("CIPHER TEXT: %d Byte + 1 Byte(Roll-over-counter)\n\n",length);
	print_hex(cipher,length+1);
	printf("\n\n");
#endif

	/* 3. Decrypt Cipher Text to decrypt text */

	decrypt_pdu(key,cipher,length+1,decrypt);

#ifdef DEBUG
	printf("DECRYPT TEXT: %d Byte\n\n",length);
	print_hex(decrypt,length);
	printf("\n\n");
```

```
#endif

	/* 4. Compare decrypt text and original plain text */

	if ( compare(decrypt,plain,length) == 0 ) {
		return 1; /* Test Success */
	} else {
		return 0; /* Test Failure */
	}
}


/*****************************************************/
/* AES CTR main()                                    */
/* Test vectors                                      */
/*****************************************************/

int main()
{
	int			i, len[] = { 64, 256, 1500 };

	for ( i=0; i<sizeof(len)/sizeof(len[0]); i++ ) {
		printf("Test %d **********************************************************\n\n",i+1);
		if ( !test_case(len[i]) ) {
			printf(" ==> Failure\n");
		}
	}
	return 0;
}
```