Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**
Title	AES in CTR mode
Date Submitted	**2005-05-01**
Source(s)	Ilan Zohar ilan.zohar@intel.com Yigal Eliaspur yigal.eliaspur@intel.com **Intel Corporation**
Re:	IEEE802.16e/D7
Abstract	This contribution clarifies use of AES in CTR and sets the ordering of the frame number.
Purpose	To incorporate the text changes proposed in this contribution into the 802.16e/D8 draft.
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>.

AES in CTR mode

Ilan Zohar (Intel corp)

1 Introduction

Section 7.8.4.1.1 prescribe how AES in CTR should be used and how the NONCE and Initial counter should be constructed. However, the endiannes of the frame-number as embedded in the NONCE and initial counter is Big Endian while in AES in CCM mode most fields are ordered as LSB first (i.e. little endian). This may confuse implementers. We therefore propose to change ordering of frame-number to little endian.
We also offer an additional small clarification to the text, and replace the test vector with vectors which include all PDU components for the benefit of implementers, thus allowing the CRC to be an inherent byte order checker. The code in this contribution uses the same encryption engine used in D7 with the necessary additions (with one modification only to the way the counter is constructed – the aforementioned frame-number endianness).

2 Proposed Text Change

[modify section 7.8.4.1.1 as follows]

7.8.4.1.1 PDU payload format

Counter mode requires a 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~~ LSbits 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.
A 32 bit nonce NONCE = n0 | n1 | n2 | n3 (n0 being the LSByte and n3 the MSByte) is made of ROC and 24bits frame number in the following way: n0=ROC and n1, n2 , n3 are the byte representation of frame-number in LSB first order. ~~The 32bit nonce made out of ROC and 24bits frame number~~
NONCE shall be repeated four times to construct the 128-bit counter block required by the AES-128 cipher. (initial counter = NONCE|NONCE|NONCE|NONCE). When incremented, this 16 byte counter will be treated as a Big Endian number.
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.
The processing yields a payload that is 8 bits longer than the plaintext payload.

[replace annex E with the following text]

E.1 Cryptographic method test vectors

E.1.1 AES CTR Mode Known Answer Test for Variable Text

E.1.1.1 TEST vector

E.1.1.1.1 Test 1

In the following examples, all the bytes appear in their storage order, starting in byte[0] on the left side and byte[1] to its right…

```
Plaintext PDU
Generic MAC Header = 00 00 46 03 ec 4e
Plaintext Payload       01 91 32 d0 96 7b 5a e6 d3 c0 2d dc b6 84 4e 04
                        18 5e 26 2b fe 73 1f 02 03 61 89 93 9b 9c 2b aa
                        74 5b 0f 9c c9 ce 86 4e e1 bb f5 ed 8b 25 77 3d
                        dd 36 c8 d8 00 00 9d 65 45 4d d8 07 61 18 ae 0f

Roll-over-counter: 1 Byte

00

PHY Synchronization (frame number): 3 Bytes

0x123456

Counter: 16 Bytes (byte 0 to byte 15)

counter =               00 56 34 12 00 56 34 12 00 56 34 12 00 56 34 12

Key (16Bytes) =         00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff

Encrypted PDU
Generic MAC Header = 40 40 4b 03 ec 8c
ROC        =            00
Ciphertext =            d9 d0 1c 4b b9 5f c7 73 eb dd d5 26 a4 20 10 34
                        85 13 9a eb ae 6c 8c 8c 2a ee cf cd 7f f8 10 6c
                        f6 9b 1e 40 24 af 17 0a ca d4 ff 8b 0a 3d 21 10
                        9c 6d 60 65 7a 55 34 55 e5 2c c9 fa 52 42 06 07
CRC =                   45 23 93 a3
```

E.1.1.1.2 Test 2

```
Plaintext PDU
Generic MAC Header = 00 01 06 9c 66 43
Plaintext Payload       47 46 97 b1 d7 ba 7d 35 bf 78 76 f3 bf 1c 9a 63
                        bd 9c 93 5d 27 3a 6d ce 85 fe c1 59 2c a9 7e 11
                        b4 82 26 f4 25 e8 b2 4e 6e 13 f8 af 28 e1 d3 96
                        31 2e d2 7a 28 82 bc 68 48 92 af c2 b9 7a 20 5f
                        d6 09 85 aa 6e 1b f4 ec 8d 59 79 60 d9 52 75 46
                        fc 4d be 92 33 c3 d7 66 81 e4 08 ff 93 0d 89 32
                        d8 a1 a9 33 d8 20 1d bf 51 f1 4a 57 24 bc d6 b6
                        9a c0 41 25 01 10 cf db 36 1e 8e 04 1e 75 c1 b0
                        8c 13 70 34 b3 4b 70 3c 94 89 a1 2a 82 f7 b3 ed
                        31 57 2e ff 76 00 19 a1 19 71 ef 0d e6 4b 3b c5
                        6a 34 a1 9b 74 78 9a a3 db d5 a0 b9 91 61 30 bd
                        8f e8 3f 31 9b b3 96 5a 7f 15 be 9d 9f b1 ce 27
                        94 df ea 9e bb 0c ab fa 51 92 4f 2d 1c df da bf
                        28 54 15 14 a5 d2 8c 73 69 4d 78 81 28 53 bd 51
                        d4 f6 e0 ba 4d f2 21 11 c9 87 9d f5 13 e1 a8 53
                        1b 82 3b 4b ec 8e aa 1e 7f 62 80 cc 83 62 b1 89

Roll-over-counter: 1 Byte

00

PHY Synchronization (frame number): 3 Bytes

0x123456

Counter: 16 Bytes (byte 0 to byte 15)

counter =               00 56 34 12 00 56 34 12 00 56 34 12 00 56 34 12

Key (16Bytes) =         00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff

Encrypted PDU
Generic MAC Header = 40 41 0b 9c 66 81
ROC        =            00
Ciphertext =            9f 07 b9 2a f8 9e e0 a0 87 65 8e 09 ad b8 c4 53
                        20 d1 2f 9d 77 25 fe 40 ac 71 87 07 c8 cd 45 d7
                        36 42 37 28 c8 89 23 0a 45 7c f2 c9 a9 f9 85 bb
                        70 75 7a c7 52 d7 15 58 e8 f3 be 3f 8a 20 88 57
                        3e 2f 4c 68 6c 86 dc bf a9 8a 27 84 b8 10 a2 d2
                        c9 d3 f3 0a 61 dd c2 4c 0c c0 a8 e1 65 0b 92 f9
                        c1 03 7f e2 84 1f 48 a7 bc 82 ce 13 14 75 f5 c8
                        13 b7 c1 31 e1 08 b6 85 55 fa 9f fc d0 e4 3c 01
                        a5 8f 8f fc 43 87 f0 2e 44 c7 d9 62 bd 57 25 df
                        c0 b2 1b 3b af ba 16 88 17 8a 89 ce 37 84 83 c0
                        00 88 34 e9 9d 18 f4 d3 53 39 27 2d 6e 03 97 4d
                        5f 75 d6 e6 56 b1 6f da 74 ab 30 55 6e e1 d5 d1
                        cd a5 37 6d 03 ac 46 2d fd c8 7e d6 3c ae f3 05
                        64 05 10 58 cc 4c f5 ed bb ad 5d 23 5a 65 f0 19
                        14 61 41 3e 08 81 41 21 69 02 1e dd 76 42 07 e7
                        d7 5e 09 1c 98 58 f4 0c 7e 75 b1 a8 27 ec 4a 8e
CRC =                   d5 91 47 ac
```

E.1.1.1.3 Test 3

```
Plaintext PDU
Generic MAC Header = 00 05 e2 bf 5d 4e
Plaintext Payload       02 b6 1a a0 dd 7e c0 80 62 ca 8e 5c f5 a2 2e 60
                        22 bf f4 9d 08 55 0f a4 22 d9 de 99 b9 db 04 21
                        b6 9e 38 11 2c a0 58 52 5f aa ce cf 87 9d cd e7
```

```
26 a2 68 41 23 db 11 6e 93 4e 8d 3a 50 1d cf 23
49 ca e6 cb bf 9e 5d dc 3b de 3b 40 8b fd 0e 15
87 6e 19 43 e5 3c 26 21 ef 15 0d 10 57 eb 67 6f
f3 d7 89 d8 b2 66 43 02 87 f5 67 3e 96 42 b0 f2
70 e4 02 f1 9a ca 94 25 39 63 02 6b 16 a9 db 11
d0 aa b2 ce 88 b0 25 b1 bb c8 08 df a5 b5 16 8e
f3 10 47 26 fc 9f 4b ed 5f b2 38 28 3e 86 e7 1c
e8 72 15 c9 2c fa c7 df 36 75 01 c1 1e 68 50 ff
0a 41 31 40 28 9f e3 f1 33 c8 a6 a9 e9 75 ec ab
27 a3 92 6d f2 8b 95 8a 42 66 5a 0a cc 33 12 64
98 13 33 28 a5 75 9b b3 71 b0 66 d6 98 34 f4 e1
65 fd 30 e6 94 72 a1 b6 0d 4a 43 69 e6 b6 bb e6
65 66 e8 50 64 93 5b bc bf bd be 24 34 46 af ea
5f 85 1d e9 34 85 a8 71 b1 18 15 14 08 5b 4f b3
25 66 12 ad ba 33 b2 9e a8 8c 1a 8c 0d fa 77 f9
bb 88 af b2 60 64 0d d0 2c 10 50 c9 37 53 7d 03
71 83 9c c3 67 5a d8 f2 a0 00 0e 91 de 64 4f 49
07 9d 65 05 08 75 dc f0 67 bb 9d cf e1 96 99 15
cc 77 98 97 92 d1 ad 58 03 45 58 3c c6 60 df 1e
ba a2 e5 2f 8a e7 c8 f8 33 e7 cd f5 da e4 a1 30
9e 44 41 bc ca 2c b7 7c 15 cf db 20 4f 93 7a c4
32 bb ff 04 a6 b2 2d 13 47 ae d7 8f 60 c8 3c a4
3c 35 f9 49 06 c5 27 0d 03 5a a5 58 6b 6b 18 8c
b4 57 a9 e1 8a 92 0e 78 43 6e de 7d 17 91 b6 c8
de da 4c de a6 c0 d5 c4 e0 e9 ea 86 71 1c 5b d3
6e 2c 01 a9 c7 11 17 63 b0 cf 28 25 0d 59 05 fb
a6 0e eb a4 6f 06 3f 65 a8 c9 06 d4 24 a3 8d fc
77 37 4d c9 58 7c 9c 1b fd c2 27 75 b7 00 c7 a5
a0 f3 b1 4b 92 4b 8d b7 40 8c 7f f4 ac c4 a0 73
cd bf fe 35 a2 e9 98 ec 82 7e 74 e4 f1 2f 41 34
bc f1 a1 0b a5 09 8f 8c 73 10 ff 21 9a 0d 2d aa
56 51 a8 6a 6e 39 ae 2b 80 82 ce 70 02 57 61 21
d4 bc e5 a5 a7 84 bc bd f6 78 5d 1d e8 d3 75 1c
df c3 0b 6b f0 12 2b 37 1e 98 1f 9f 95 b1 bc eb
ac 4d d0 61 00 c6 35 2c 64 2f 95 33 f6 31 62 4e
1f 33 0c c6 c3 e2 01 73 6e ce 77 7f c0 3e 90 18
ed e5 dc 10 7e a2 c0 c2 42 e9 cb 34 8e 0e 86 cb
b5 07 bc 8e eb e1 cd 4f 66 7c 0e a7 01 c5 c2 39
28 0f ac 09 59 b3 cd 70 fc a3 4c 7b e0 15 1a 25
25 ec 50 60 d1 0c d1 3e e6 42 45 36 3b d9 e0 e4
d9 9c 0b 2d 30 5c 74 30 e5 a0 8c ea 86 bc 00 f9
df d8 26 60 49 2c fa be b8 08 a6 d1 bd d2 20 b9
61 a7 eb e4 09 c7 73 02 3b 6c 29 ec 81 3f c2 e9
37 09 c6 3b 8f cf db 55 8a fe df a6 3b d1 5f 5e
09 90 66 1f 54 e7 34 f1 21 da e5 73 39 a3 90 a0
6d 05 de 24 45 4b af 90 f6 9c ca 6d d2 be 23 85
06 02 c0 56 e7 77 c5 0d a2 07 af f9 81 b5 45 d3
```

```
a6 99 44 d9 76 c0 5b 04 7a a1 68 63 08 4a 99 e3
6f ef 63 89 03 fa de 70 b4 56 9b 80 90 0a 61 3c
f0 dd f7 9c 96 14 69 4f 80 15 e0 4e c9 ef 97 38
48 90 e0 41 4e bb de 3c 31 73 e4 94 08 ff 10 9f
42 2c 1e 3c 7f f6 0d 15 57 48 84 7e 6a ed 9a 4b
7a 67 f4 8d d6 cc ce 98 df 52 f1 45 f2 b9 21 c7
7a 2e 07 0b 75 dd 23 04 37 d2 cd c8 a9 4d c9 ec
d8 3f 80 06 14 08 5c b8 69 2f 4d 2f bf 23 0f 87
5c cf 29 e7 22 08 2f d2 40 93 59 8b ab df ef f2
1b 28 8f cd e6 13 e0 d3 64 8d ad 74 49 f4 fc bb
99 46 22 33 9a 7d 5b 3b 7b b2 f3 ac fc 3d 84 3d
e6 7c 54 8a 93 57 5a 2c 4b 39 ed be cf a7 b0 46
c5 11 b9 dc 58 0c 7c 06 d7 9f 8b 9b 92 c8 a4 b3
c3 df 29 6d ca 06 71 0b 82 45 12 3f fc 82 9e 13
5f f5 dd 57 3e 4b 0e ff 2d 12 39 4e ca a7 16 44
23 38 91 2e a2 1c 75 c3 56 0f 49 b2 e0 92 df 15
cc 00 a3 9f 99 98 32 fb 3c 0b 3e 42 69 cc 46 e7
62 bb 35 0e 9b 5b 5d ac fa 3a e6 59 f5 ab 34 49
5b 89 4b 39 1a 1c b6 d9 ac d2 02 7e 9b f0 4b 9c
c0 e0 ea d5 9c 4f c9 27 8b b1 66 ff 1b 6a 08 b0
43 2c 3b 31 dd c4 0d 7c 0f f7 16 91 f7 93 e1 69
68 6a a9 d3 f2 49 01 9e 0d a8 6d f5 9b 34 6b 58
a0 cf 02 1b 65 47 52 d4 db 50 33 91 79 ff 6f 5f
6c 63 95 e1 57 65 f4 83 6c 9c c7 16 27 36 17 52
79 a2 56 15 9e 25 48 d4 71 ff 39 1c 85 44 02 94
c6 1c f9 5f e8 86 3a 4e 7b 50 6b c4 d8 64 6e ba
bd 18 15 c1 d9 a4 5d 7a 19 6b 31 58 ea 3b 50 28
58 2f 46 35 2d 56 11 81 f7 d0 73 ea 2e 7c 79 b4
40 f0 48 4d d3 d2 a2 cc 02 43 4b f4 dd 85 b6 42
ec 7e 18 d4 14 49 63 a5 84 6f 25 fa 14 02 eb 6a
c2 31 1b 6d ae 88 d3 d7 46 80 df 27 f9 8a 39 12
35 33 32 34 f5 98 be 4c af ca ec ee d9 43 1b 10
e8 26 e5 5d f5 62 55 b0 e4 62 6f ad 45 7c 88 ce
cc c0 7d d4 90 48 59 0e ea c4 5f 49 37 54 0d e2
40 68 26 e1 9c c9 33 74 c4 6f a3 ce 2d 53 f7 b6
33 dd 0d bf 0a 22 16 8e 92 89 39 14 4e 11 6a 23
41 d3 83 45 fd eb 20 49 b5 79 4c 59 87 d0 85 12
d5 8e 1b 83 f3 ba 7a 74 ec 8d 5e e3 a9 20 82 1d
48 8c cb 5f dc bf 77 5d 72 96 62 a2 8f 7c d7 30
03 1c 0b 39 42 6a b6 72 23 8c dc cf 38 ed 51 25
9c 03 f5 8a 64 04 3d e3 98 28 05 8a eb a8 3a 68
f8 19 67 82 56 55 a1 41 4b 89 e7 7b 55 ad 76 96
6a ed 1f ab 27 40 1d 1b b1 d4 7e 75 a8 6c a4 1a
d5 5f e1 86 f7 66 ba a2 60 d1 db 11
```

Roll-over-counter: 1 Byte

```
00

PHY Synchronization (frame number): 3 Bytes

0x123456

Counter: 16 Bytes (byte 0 to byte 15)

counter =               00 56 34 12 00 56 34 12 00 56 34 12 00 56 34 12

Key (16Bytes) =         00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff

Encrypted PDU
Generic MAC Header = 40 45 e7 bf 5d dd
ROC          =          00
Ciphertext =            da f7 34 3b f2 5a 5d 15 5a d7 76 a6 e7 06 70 50
                        bf f2 48 5d 58 4a 9c 2a 0b 56 98 c7 5d bf 3f e7
                        34 5e 29 cd c1 c1 c9 16 74 c5 c4 a9 06 85 9b ca
                        67 f9 c0 fc 59 8e b8 5e 33 2f 9c c7 63 47 67 2b
                        a1 ec 2f 09 bd 03 75 8f 1f 0d 65 a4 ea bf d9 81
                        b2 f0 54 db b7 22 33 0b 62 31 ad 0e a1 ed 7c a4
                        ea 75 5f 09 ee 59 16 1a 6a 86 e3 7a a6 8b 93 8c
                        f9 93 82 e5 7a d2 ed 7b 5a 87 13 93 d8 38 26 a0
                        f9 36 4d 06 78 7c a5 a3 6b 86 70 97 9a 15 80 bc
                        02 f5 72 e2 25 25 44 c4 51 49 5e eb ef 49 5f 19
                        82 ce 80 bb c5 9a a9 af be 99 86 55 e1 0a f7 0f
                        da dc d8 97 e5 9d 1a 71 38 76 28 61 18 25 f7 5d
                        7e d9 4f 9e 4a 2b 78 5d ee 3c 6b f1 ec 42 3b de
                        d4 42 36 64 cc eb e2 2d a3 50 43 74 ea 02 b9 a9
                        a5 6a 91 62 d1 01 c1 86 ad cf c0 41 83 15 14 52
                        a9 ba da 07 10 45 05 ae be aa 8f 40 90 c8 54 ed
                        5c 84 f1 d5 f5 f1 4c 29 19 c9 51 06 35 9b f2 ec
                        56 13 e2 b3 25 22 cd 23 be 23 0a 52 cb 62 4b 44
                        40 82 c7 e1 12 c6 49 07 81 a7 6d e2 80 fb 45 87
                        65 02 c2 95 74 fa 4f bc af 23 0e df 9c dc eb 95
                        75 60 8e c4 50 7e c7 f6 52 8e 68 56 d9 18 a7 3a
                        f9 11 44 4b b4 ea 9d 5b 82 f3 a6 a3 ee e0 31 c4
                        3b 52 67 65 81 73 f6 d5 e3 06 a9 85 43 38 ab fd
                        38 03 e3 aa 6f 52 fc 6c a4 aa 5f 45 6e 25 8e 9c
                        4e 0c cb 46 ff c8 0a a5 4c 68 7f 42 8b 8b 38 78
                        21 17 b7 7a b9 09 e6 13 0e d2 f1 a1 79 1d 22 36
                        df 97 53 f9 5f 98 00 98 ed be 2f cc 54 c2 ae bd
                        34 07 98 c0 f0 f2 d4 d7 1a 0b 59 d6 f0 e3 cb c7
                        4b ab 1b b4 40 6b 25 a8 11 ea 08 74 97 e0 ac 28
                        1d e2 fd 13 a0 6d 7a fa f6 69 85 7a e3 24 c5 a3
                        95 e1 67 ba 4e 28 ca 01 95 b8 7a 3f 54 a9 a2 78
                        1c e0 54 95 86 22 95 4e 8b cd 93 59 20 df d3 8f
```

```
e0 7e fe df 43 b8 f9 22 2e 52 a7 5c d1 53 69 ce
96 b6 19 72 4b 69 58 04 3d 05 33 f5 9d 42 89 62
87 61 25 bc ab 23 e0 4b 15 1b f0 26 2a 0e 56 1a
dc 52 21 43 78 43 ab b6 52 f1 20 95 57 d6 ec 78
12 e9 5d 40 d3 c1 e5 92 f2 f1 0c e4 99 70 39 49
73 a8 90 da 1d 0f 61 0a 0c f4 e7 d0 56 a0 79 81
d1 14 25 f0 ce 6e 2e 96 c6 ee c0 3d cd 88 88 d7
2f 6f cb 52 51 01 37 71 c4 05 e2 2d c1 33 08 26
10 81 c4 fc 8e fa 26 79 e7 52 b6 09 fb 1f c9 5a
bb e8 db 1e 00 1c d6 65 c4 17 b0 01 85 b2 76 7c
b6 ae 41 29 9e 5d 35 c4 42 35 6d 91 e2 85 d0 a6
52 75 4f 17 6d 13 23 10 be 85 2f 83 2e 8c 3a fa
42 8f 91 36 83 49 87 22 98 02 2b c4 a6 4e 0c d2
28 8a 91 b1 69 c3 fc 50 8e 82 be c1 41 c5 92 4d
66 f0 e7 dd 35 fa 5e 91 20 84 d4 4e 3a fa 9f 7f
29 2b 39 fc 63 3d 6f a2 16 93 81 ca 73 f0 e6 3f
8c 76 15 a6 f1 e6 d0 f2 75 f8 5d 07 cd a6 7e 0c
a8 8b 5b 49 9c 43 ae 9f 67 d4 51 9e 45 fb 11 98
ba dd 22 58 0d b7 e8 dd ef 4c 96 72 33 0b 33 72
d1 2a 02 08 7c 6a 83 60 65 4c 1d 71 04 9e e4 6f
c0 70 d2 6e 46 5a d8 34 ac 75 5f c5 a7 e9 c1 53
ca 4d 7e f9 fb 81 d3 05 5a be b9 a4 59 e8 47 c6
2c 50 f6 6e 64 21 98 38 73 36 05 58 99 ca a3 ab
30 01 9c 01 7a a0 d2 67 07 67 ea d9 63 d1 a0 ae
2e d3 72 6c 09 7a ef 15 f1 b6 fe 4f 90 3b f9 d9
2a fc 4e 8f 19 32 6d 2f 9a 00 90 4b 7e 48 c5 f3
b8 60 43 70 b3 97 96 89 25 2e c4 f9 9f 9e 30 53
91 40 73 96 f6 a9 89 c0 d7 6f 25 63 38 94 17 66
7b 8b 30 2b b4 9a 94 07 95 ca 2d 06 b9 39 27 4b
cc a1 d5 96 81 60 df 60 1e e6 0c 5f 74 62 b1 e6
9c e8 c0 dc 1b 02 ac d2 35 22 c9 07 65 0a 68 b3
0d be ce 64 cd 73 57 a9 67 d0 d6 bf 4a 44 61 3f
67 17 17 86 f0 f6 67 d9 3d 76 34 c6 27 cb 73 e7
f3 ae 14 17 18 ac 7f 60 56 d3 6c 04 77 d3 2a 7c
5d 6d cf b0 8b d3 d9 b0 18 a6 9a 76 69 9c f7 4d
35 51 c8 b1 84 95 8f 49 28 ea 94 31 ea 97 76 69
18 b6 b4 84 57 70 ac ed a6 0d 7e 03 db 24 17 51
b9 57 e7 d0 28 af 3c db e5 5d 80 ff 26 02 5f 49
6c 44 f2 30 de 19 4e e3 a0 9f cf 71 26 95 53 87
37 1e 69 4a 7f a2 fc 6e 11 22 32 44 e2 b5 c2 e6
f1 8a 84 f7 30 61 7d e6 60 05 ed eb b6 f1 86 28
23 8a 30 53 e2 75 f2 74 ee 2d c0 90 54 e4 1e cc
1b 47 72 31 ae e4 3a 4c 31 f1 28 71 91 34 2c 21
1a a3 9e aa 3c fa 10 fe 99 c5 e7 5c 68 03 f6 2f
36 07 52 36 8b 88 99 62 bd 96 e9 38 66 b5 67 1b
b3 57 b4 bf 13 7c 1e 8d 48 22 f6 47 46 e9 de 2c
cc b4 2b 73 59 3c bb e7 e9 d9 9a 57 84 43 e9 53
```

```
                        89 59 30 1d 46 c7 d8 f6 ed 36 ec 7c c0 fa d2 cd
                        ad d1 1d 7a 53 05 03 f3 3f 90 ab 52 56 8e 4e 1b
                        63 b0 c5 84 e6 7c 52 eb d8 7a 09 1c 28 99 f0 ca
                        38 19 5a 4e fe b2 68 62 7c fb 23 df 61 f0 48 13
                        53 fc 53 29 36 00 03 c9 f1 ba a7 7c 66 78 00 4b
                        11 35 bb 32 55 75 e7 e2 ab 19 36 2d 17 9b 0b 04
                        09 a2 d6 4c 7d 0b f1 bf 3e b6 47 65 f6 fa 06 bf
                        de d6 a9 89 2c f8 01 29 a8 12 1d 46 13 0e 80 90
                        ce 35 f4 1f e9 72 59 9e 7b 26 b0 46 37 06 d7 c5
                        fe 5d 16 de 20 70 c1 3b 12 49 77 ea df 69 a1 fc
                        2f d6 2d 6e a7 44 f4 36 14 5a 49 41 9b 33 06 c1
                        fc c7 6d 29 dd ee 6b 88 ff f0 b0 65 de 5c 32 5e
                        e6 d9 11 e7 95 b3 dd e7 d1 a6 74 00 bc 77 4f 8a
                        1d 5d d4 fa 49 f2 95 1e b7 e3 40 77 ae b0 b9 69
                        2c 0e 6a 66 c9 24 62 88 71 6b 0e b4
CRC  =                  1c 95 b9 80
```

E.1.1.2 TEST code

```
/*****************************************************************/
/* 802.16e MBS (Multimedia Broadcast Service) AES-CTR mode example      */
/* program for KAT (Known Answer Test). KAT help implementers to        */
/* verify AES algorithm and CTR mode correctly for MBS defined          */
/* in PKMv2                                                             */
/* Version Number: 0.5                                                  */
/* The code relies on code written by JunHyuk Song, Jicheol Lee         */
/* DJ Johnston and Ilan Zohar                                           */
/*****************************************************************/
#include <stdlib.h>
#include <stdio.h>
#define MAX_BUF 10000
#define DEBUG
/*****************************/
/*** 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
};


/***************************************/
/********* Table for CRC 32 ************/
/***************************************/

 unsigned char lookahead_table[1024] =
 {
     0x00,0x00,0x00,0x00,0x96,0x30,0x07,0x77,0x2c,0x61,0x0e,0xee,0xba,0x51,0x09,0x99,
     0x19,0xc4,0x6d,0x07,0x8f,0xf4,0x6a,0x70,0x35,0xa5,0x63,0xe9,0xa3,0x95,0x64,0x9e,
     0x32,0x88,0xdb,0x0e,0xa4,0xb8,0xdc,0x79,0x1e,0xe9,0xd5,0xe0,0x88,0xd9,0xd2,0x97,
     0x2b,0x4c,0xb6,0x09,0xbd,0x7c,0xb1,0x7e,0x07,0x2d,0xb8,0xe7,0x91,0x1d,0xbf,0x90,
     0x64,0x10,0xb7,0x1d,0xf2,0x20,0xb0,0x6a,0x48,0x71,0xb9,0xf3,0xde,0x41,0xbe,0x84,
     0x7d,0xd4,0xda,0x1a,0xeb,0xe4,0xdd,0x6d,0x51,0xb5,0xd4,0xf4,0xc7,0x85,0xd3,0x83,
     0x56,0x98,0x6c,0x13,0xc0,0xa8,0x6b,0x64,0x7a,0xf9,0x62,0xfd,0xec,0xc9,0x65,0x8a,
     0x4f,0x5c,0x01,0x14,0xd9,0x6c,0x06,0x63,0x63,0x3d,0x0f,0xfa,0xf5,0x0d,0x08,0x8d,
     0xc8,0x20,0x6e,0x3b,0x5e,0x10,0x69,0x4c,0xe4,0x41,0x60,0xd5,0x72,0x71,0x67,0xa2,
     0xd1,0xe4,0x03,0x3c,0x47,0xd4,0x04,0x4b,0xfd,0x85,0x0d,0xd2,0x6b,0xb5,0x0a,0xa5,
     0xfa,0xa8,0xb5,0x35,0x6c,0x98,0xb2,0x42,0xd6,0xc9,0xbb,0xdb,0x40,0xf9,0xbc,0xac,
     0xe3,0x6c,0xd8,0x32,0x75,0x5c,0xdf,0x45,0xcf,0x0d,0xd6,0xdc,0x59,0x3d,0xd1,0xab,
     0xac,0x30,0xd9,0x26,0x3a,0x00,0xde,0x51,0x80,0x51,0xd7,0xc8,0x16,0x61,0xd0,0xbf,
     0xb5,0xf4,0xb4,0x21,0x23,0xc4,0xb3,0x56,0x99,0x95,0xba,0xcf,0x0f,0xa5,0xbd,0xb8,
     0x9e,0xb8,0x02,0x28,0x08,0x88,0x05,0x5f,0xb2,0xd9,0x0c,0xc6,0x24,0xe9,0x0b,0xb1,
     0x87,0x7c,0x6f,0x2f,0x11,0x4c,0x68,0x58,0xab,0x1d,0x61,0xc1,0x3d,0x2d,0x66,0xb6,
     0x90,0x41,0xdc,0x76,0x06,0x71,0xdb,0x01,0xbc,0x20,0xd2,0x98,0x2a,0x10,0xd5,0xef,
     0x89,0x85,0xb1,0x71,0x1f,0xb5,0xb6,0x06,0xa5,0xe4,0xbf,0x9f,0x33,0xd4,0xb8,0xe8,
     0xa2,0xc9,0x07,0x78,0x34,0xf9,0x00,0x0f,0x8e,0xa8,0x09,0x96,0x18,0x98,0x0e,0xe1,
     0xbb,0x0d,0x6a,0x7f,0x2d,0x3d,0x6d,0x08,0x97,0x6c,0x64,0x91,0x01,0x5c,0x63,0xe6,
     0xf4,0x51,0x6b,0x6b,0x62,0x61,0x6c,0x1c,0xd8,0x30,0x65,0x85,0x4e,0x00,0x62,0xf2,
     0xed,0x95,0x06,0x6c,0x7b,0xa5,0x01,0x1b,0xc1,0xf4,0x08,0x82,0x57,0xc4,0x0f,0xf5,
     0xc6,0xd9,0xb0,0x65,0x50,0xe9,0xb7,0x12,0xea,0xb8,0xbe,0x8b,0x7c,0x88,0xb9,0xfc,
     0xdf,0x1d,0xdd,0x62,0x49,0x2d,0xda,0x15,0xf3,0x7c,0xd3,0x8c,0x65,0x4c,0xd4,0xfb,
     0x58,0x61,0xb2,0x4d,0xce,0x51,0xb5,0x3a,0x74,0x00,0xbc,0xa3,0xe2,0x30,0xbb,0xd4,
     0x41,0xa5,0xdf,0x4a,0xd7,0x95,0xd8,0x3d,0x6d,0xc4,0xd1,0xa4,0xfb,0xf4,0xd6,0xd3,
     0x6a,0xe9,0x69,0x43,0xfc,0xd9,0x6e,0x34,0x46,0x88,0x67,0xad,0xd0,0xb8,0x60,0xda,
     0x73,0x2d,0x04,0x44,0xe5,0x1d,0x03,0x33,0x5f,0x4c,0x0a,0xaa,0xc9,0x7c,0x0d,0xdd,
     0x3c,0x71,0x05,0x50,0xaa,0x41,0x02,0x27,0x10,0x10,0x0b,0xbe,0x86,0x20,0x0c,0xc9,
     0x25,0xb5,0x68,0x57,0xb3,0x85,0x6f,0x20,0x09,0xd4,0x66,0xb9,0x9f,0xe4,0x61,0xce,
     0x0e,0xf9,0xde,0x5e,0x98,0xc9,0xd9,0x29,0x22,0x98,0xd0,0xb0,0xb4,0xa8,0xd7,0xc7,
     0x17,0x3d,0xb3,0x59,0x81,0x0d,0xb4,0x2e,0x3b,0x5c,0xbd,0xb7,0xad,0x6c,0xba,0xc0,
     0x20,0x83,0xb8,0xed,0xb6,0xb3,0xbf,0x9a,0x0c,0xe2,0xb6,0x03,0x9a,0xd2,0xb1,0x74,
     0x39,0x47,0xd5,0xea,0xaf,0x77,0xd2,0x9d,0x15,0x26,0xdb,0x04,0x83,0x16,0xdc,0x73,
     0x12,0x0b,0x63,0xe3,0x84,0x3b,0x64,0x94,0x3e,0x6a,0x6d,0x0d,0xa8,0x5a,0x6a,0x7a,
     0x0b,0xcf,0x0e,0xe4,0x9d,0xff,0x09,0x93,0x27,0xae,0x00,0x0a,0xb1,0x9e,0x07,0x7d,
     0x44,0x93,0x0f,0xf0,0xd2,0xa3,0x08,0x87,0x68,0xf2,0x01,0x1e,0xfe,0xc2,0x06,0x69,
     0x5d,0x57,0x62,0xf7,0xcb,0x67,0x65,0x80,0x71,0x36,0x6c,0x19,0xe7,0x06,0x6b,0x6e,
     0x76,0x1b,0xd4,0xfe,0xe0,0x2b,0xd3,0x89,0x5a,0x7a,0xda,0x10,0xcc,0x4a,0xdd,0x67,
     0x6f,0xdf,0xb9,0xf9,0xf9,0xef,0xbe,0x8e,0x43,0xbe,0xb7,0x17,0xd5,0x8e,0xb0,0x60,
     0xe8,0xa3,0xd6,0xd6,0x7e,0x93,0xd1,0xa1,0xc4,0xc2,0xd8,0x38,0x52,0xf2,0xdf,0x4f,
     0xf1,0x67,0xbb,0xd1,0x67,0x57,0xbc,0xa6,0xdd,0x06,0xb5,0x3f,0x4b,0x36,0xb2,0x48,
     0xda,0x2b,0x0d,0xd8,0x4c,0x1b,0x0a,0xaf,0xf6,0x4a,0x03,0x36,0x60,0x7a,0x04,0x41,
     0xc3,0xef,0x60,0xdf,0x55,0xdf,0x67,0xa8,0xef,0x8e,0x6e,0x31,0x79,0xbe,0x69,0x46,
     0x8c,0xb3,0x61,0xcb,0x1a,0x83,0x66,0xbc,0xa0,0xd2,0x6f,0x25,0x36,0xe2,0x68,0x52,
     0x95,0x77,0x0c,0xcc,0x03,0x47,0x0b,0xbb,0xb9,0x16,0x02,0x22,0x2f,0x26,0x05,0x55,
     0xbe,0x3b,0xba,0xc5,0x28,0x0b,0xbd,0xb2,0x92,0x5a,0xb4,0x2b,0x04,0x6a,0xb3,0x5c,
     0xa7,0xff,0xd7,0xc2,0x31,0xcf,0xd0,0xb5,0x8b,0x9e,0xd9,0x2c,0x1d,0xae,0xde,0x5b,
     0xb0,0xc2,0x64,0x9b,0x26,0xf2,0x63,0xec,0x9c,0xa3,0x6a,0x75,0x0a,0x93,0x6d,0x02,
     0xa9,0x06,0x09,0x9c,0x3f,0x36,0x0e,0xeb,0x85,0x67,0x07,0x72,0x13,0x57,0x00,0x05,
     0x82,0x4a,0xbf,0x95,0x14,0x7a,0xb8,0xe2,0xae,0x2b,0xb1,0x7b,0x38,0x1b,0xb6,0x0c,
     0x9b,0x8e,0xd2,0x92,0x0d,0xbe,0xd5,0xe5,0xb7,0xef,0xdc,0x7c,0x21,0xdf,0xdb,0x0b,
     0xd4,0xd2,0xd3,0x86,0x42,0xe2,0xd4,0xf1,0xf8,0xb3,0xdd,0x68,0x6e,0x83,0xda,0x1f,
     0xcd,0x16,0xbe,0x81,0x5b,0x26,0xb9,0xf6,0xe1,0x77,0xb0,0x6f,0x77,0x47,0xb7,0x18,
     0xe6,0x5a,0x08,0x88,0x70,0x6a,0x0f,0xff,0xca,0x3b,0x06,0x66,0x5c,0x0b,0x01,0x11,
     0xff,0x9e,0x65,0x8f,0x69,0xae,0x62,0xf8,0xd3,0xff,0x6b,0x61,0x45,0xcf,0x6c,0x16,
     0x78,0xe2,0x0a,0xa0,0xee,0xd2,0x0d,0xd7,0x54,0x83,0x04,0x4e,0xc2,0xb3,0x03,0x39,
     0x61,0x26,0x67,0xa7,0xf7,0x16,0x60,0xd0,0x4d,0x47,0x69,0x49,0xdb,0x77,0x6e,0x3e,
     0x4a,0x6a,0xd1,0xae,0xdc,0x5a,0xd6,0xd9,0x66,0x0b,0xdf,0x40,0xf0,0x3b,0xd8,0x37,
     0x53,0xae,0xbc,0xa9,0xc5,0x9e,0xbb,0xde,0x7f,0xcf,0xb2,0x47,0xe9,0xff,0xb5,0x30,
     0x1c,0xf2,0xbd,0xbd,0x8a,0xc2,0xba,0xca,0x30,0x93,0xb3,0x53,0xa6,0xa3,0xb4,0x24,
     0x05,0x36,0xd0,0xba,0x93,0x06,0xd7,0xcd,0x29,0x57,0xde,0x54,0xbf,0x67,0xd9,0x23,
      0x2e,0x7a,0x66,0xb3,0xb8,0x4a,0x61,0xc4,0x02,0x1b,0x68,0x5d,0x94,0x2b,0x6f,0x2a,
      0x37,0xbe,0x0b,0xb4,0xa1,0x8e,0x0c,0xc3,0x1b,0xdf,0x05,0x5a,0x8d,0xef,0x02,0x2d

  };

/****************************************/
/********* Table for CRC 8  ************/
/****************************************/

 unsigned char crc8_lookahead_table[256] =
 {
  0x00, 0x07, 0x0E, 0x09, 0x1C, 0x1B, 0x12, 0x15, 0x38, 0x3F, 0x36, 0x31, 0x24, 0x23, 0x2A, 0x2D,
  0x70, 0x77, 0x7E, 0x79, 0x6C, 0x6B, 0x62, 0x65, 0x48, 0x4F, 0x46, 0x41, 0x54, 0x53, 0x5A, 0x5D,
  0xE0, 0xE7, 0xEE, 0xE9, 0xFC, 0xFB, 0xF2, 0xF5, 0xD8, 0xDF, 0xD6, 0xD1, 0xC4, 0xC3, 0xCA, 0xCD,
  0x90, 0x97, 0x9E, 0x99, 0x8C, 0x8B, 0x82, 0x85, 0xA8, 0xAF, 0xA6, 0xA1, 0xB4, 0xB3, 0xBA, 0xBD,
  0xC7, 0xC0, 0xC9, 0xCE, 0xDB, 0xDC, 0xD5, 0xD2, 0xFF, 0xF8, 0xF1, 0xF6, 0xE3, 0xE4, 0xED, 0xEA,
  0xB7, 0xB0, 0xB9, 0xBE, 0xAB, 0xAC, 0xA5, 0xA2, 0x8F, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9D, 0x9A,
  0x27, 0x20, 0x29, 0x2E, 0x3B, 0x3C, 0x35, 0x32, 0x1F, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0D, 0x0A,
  0x57, 0x50, 0x59, 0x5E, 0x4B, 0x4C, 0x45, 0x42, 0x6F, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7D, 0x7A,
  0x89, 0x8E, 0x87, 0x80, 0x95, 0x92, 0x9B, 0x9C, 0xB1, 0xB6, 0xBF, 0xB8, 0xAD, 0xAA, 0xA3, 0xA4,
  0xF9, 0xFE, 0xF7, 0xF0, 0xE5, 0xE2, 0xEB, 0xEC, 0xC1, 0xC6, 0xCF, 0xC8, 0xDD, 0xDA, 0xD3, 0xD4,
  0x69, 0x6E, 0x67, 0x60, 0x75, 0x72, 0x7B, 0x7C, 0x51, 0x56, 0x5F, 0x58, 0x4D, 0x4A, 0x43, 0x44,
  0x19, 0x1E, 0x17, 0x10, 0x05, 0x02, 0x0B, 0x0C, 0x21, 0x26, 0x2F, 0x28, 0x3D, 0x3A, 0x33, 0x34,
  0x4E, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5C, 0x5B, 0x76, 0x71, 0x78, 0x7F, 0x6A, 0x6D, 0x64, 0x63,
  0x3E, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2C, 0x2B, 0x06, 0x01, 0x08, 0x0F, 0x1A, 0x1D, 0x14, 0x13,
  0xAE, 0xA9, 0xA0, 0xA7, 0xB2, 0xB5, 0xBC, 0xBB, 0x96, 0x91, 0x98, 0x9F, 0x8A, 0x8D, 0x84, 0x83,
  0xDE, 0xD9, 0xD0, 0xD7, 0xC2, 0xC5, 0xCC, 0xCB, 0xE6, 0xE1, 0xE8, 0xEF, 0xFA, 0xFD, 0xF4, 0xF3
 };

/******************************/
/**** Function Prototypes ****/
/******************************/
void bitwise_xor(unsigned char *ina, unsigned char *inb, unsigned char *out);
void blockprint_payload(unsigned char *str, unsigned char *payload, int length);

/****************************************************************/
/************** 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();
        }
}

/****************************************/
/* CRC8()                               */
/* Calculates the CRC8  of a sequence   */
/* of octets.                           */
/****************************************/

void crc8(unsigned char *crc, unsigned char *data, int length)
{
   int i;
   int index;
   unsigned char ch;
//    unsigned char table_entry;
```

```
    *crc = 0x00;
    for (i=0; i<length; i++)
    {
        ch = data[i];
        index = (((int)((*crc) ^ ch)) & 0xff);
        *crc = crc8_lookahead_table[index];
    }
}

/****************************************/
/* CRC32()                              */
/* Calculates the CRC32 of a sequence   */
/* of octets.                           */
/****************************************/

void crc32(unsigned char *crc, unsigned char *data, int length)
{

    int i;
    int index;
    unsigned char ch;
    unsigned char table_entry[4];

    crc[3] = 0xff;
    crc[2] = 0xff;
    crc[1] = 0xff;
    crc[0] = 0xff;

    for (i=0; i<length; i++)
    {
        ch = data[i];

        index = (((int)(crc[0] ^ ch)) & 0xff) * 4;

        table_entry[0] = lookahead_table[index];
        table_entry[1] = lookahead_table[index+1];
        table_entry[2] = lookahead_table[index+2];
        table_entry[3] = lookahead_table[index+3];

        crc[0] = crc[1] ^ table_entry[0];
        crc[1] = crc[2] ^ table_entry[1];
        crc[2] = crc[3] ^ table_entry[2];
        crc[3] = table_entry[3];

    }
    crc[0] = crc[0] ^ 0xff;
    crc[1] = crc[1] ^ 0xff;
    crc[2] = crc[2] ^ 0xff;
    crc[3] = crc[3] ^ 0xff;
}



/****************************************************************************/
/* 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 (aka Nonce) by concatenating 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;

        for ( i=0; i<4; i++ ) { /* 4 copies of the nonce */
                ctr[i*4+0] = rollcnt;
                ctr[i*4+1] = (unsigned char)(phy_sync & 0xff);
                ctr[i*4+2] = (unsigned char)((phy_sync >> 8 ) & 0xff);
                ctr[i*4+3] = (unsigned char)((phy_sync >> 16 ) & 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 field is 123456 in this example. */
        return 0x00123456;
}



/********************************************************/
/* 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 Synchronization (frame number): 3 Bytes\n\n");
        printf("0x%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 (byte 0 to byte 15)\n\n");
        blockprint_payload("counter =         ",ctr,16);
        printf("\n");
        blockprint_payload("Key (16Bytes) = ", 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;
}


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;
}
```

```
/*
 *       PlainText should be placed starting byte 6
 *  Generated PDU has no CRC protection at the end
 */
int set_gmh_for_plain(unsigned char* pdu, int lengthOfPlainText)
{
        pdu[0] = 0x00; // unencrypted pdu
        pdu[1] = 0x00; // no crc protection
        pdu[1] |= (((lengthOfPlainText + 6) /256 ) % 256 ) & 0x07; // setting 3 MSB bits of PDU length
        pdu[2] = (unsigned char)( (lengthOfPlainText + 6 ) %256);  //setting 8 MSB bits of PDU length
        pdu[3] = random_8bit(); // bogus CID MSB
        pdu[4] = random_8bit(); // bogus CID LSB
        pdu[5] = 0x00; // not really needed
        crc8(&pdu[5], pdu, 5); // compute HCS

        return lengthOfPlainText +6;
}

/*
 *       ROC should be placed in byte 6 and CipherText should be place begining byte 7
 *  lengthOfCipherTExt includes Text and ROC
 */
int set_gmh_crc_for_encrypted(unsigned char* pdu, int lengthOfCipherText,
                                                        unsigned char MSBCID, unsigned char LSBCID)
{
        int lenOfPDU = lengthOfCipherText + 6 +4;// 6 for the GMH and 4 for the CRC
        int GMHandCTlen = lengthOfCipherText + 6;
        pdu[0] = 0x40; // encrypted PDU
        pdu[1] = 0x40; // crc protection offered
        pdu[1] |= (( lenOfPDU /256 ) % 256 ) & 0x07; // setting 3 MSB bits of PDU length
        pdu[2] = (unsigned char)( lenOfPDU  %256);  //setting 8 MSB bits of PDU length
        pdu[3] = MSBCID;
        pdu[4] = LSBCID;
        pdu[5] = 0x00; // not really needed
        crc8(&pdu[5], pdu, 5); // compute HCS
   crc32(&pdu[GMHandCTlen], pdu, GMHandCTlen); // compute CRC over GMH and Cipher Text
        return lenOfPDU;
}

void blockprint_gmh(unsigned char *str, unsigned char *gmh)
{
        printf("%s = %02x %02x %02x %02x %02x %02x\n", str,
                gmh[0], gmh[1], gmh[2], gmh[3], gmh[4], gmh[5]);
}

void blockprint_payload(unsigned char *str, unsigned char *payload, int length)
```

```
{
    int blocks;
    int residue;
    int i;
    int j;
    unsigned char *ptr;

    ptr = payload;
    blocks = length/16;
    residue = length % 16;

    printf("%s",str);
    if (blocks > 0)
    {
        printf ("\t");
        for (j=0;j<15;j++)
        {
            printf("%02x ",*ptr++);
        }
        printf("%02x\n",*ptr++);
    }

    for (i=1;i<blocks;i++)
    {
        printf("\t\t\t");
        for (j=0;j<15;j++)
        {
            printf("%02x ",*ptr++);
        }
        printf("%02x\n",*ptr++);
    }

    if (residue > 0)
    {
        if (blocks != 0)
        {
            printf("\t\t\t");
        }
        else printf("\t");
        for(i=0;i<(residue-1);i++)
        {
            printf("%02x ",*ptr++);
        }
        printf("%02x\n",*ptr++);
    }
}
```

```
int test_case(int lengthOfPlainText)
{
        unsigned char key[16];
        unsigned char plainPDU[MAX_BUF];
        unsigned char encryptedPDU[MAX_BUF+4];
        unsigned char decrypt[MAX_BUF];

        int lengthOfPTPDU, CTlen, lengthOfCTPDU;

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

        /* 1. Generate Plain Text with length */
//      generate_plain(plain,length);
        generate_plain(&plainPDU[6],lengthOfPlainText);
        lengthOfPTPDU = set_gmh_for_plain(plainPDU, lengthOfPlainText);

#ifdef DEBUG
        printf("Plaintext PDU\n");
   blockprint_gmh("Generic MAC Header",plainPDU);
        blockprint_payload("Plaintext Payload", &plainPDU[6], lengthOfPlainText);
        printf("\n");
#endif


        /* 2. Encrypt Plain Text to Cipher Text */
        CTlen = encrypt_pdu(key,&plainPDU[6],lengthOfPlainText,&encryptedPDU[6]);
   lengthOfCTPDU = set_gmh_crc_for_encrypted(encryptedPDU, CTlen, plainPDU[3], plainPDU[4]);

#ifdef DEBUG
        printf("Encrypted PDU\n");
   blockprint_gmh("Generic MAC Header",encryptedPDU);
        blockprint_payload("ROC        =     ", &encryptedPDU[6], 1);
        blockprint_payload("Ciphertext =     ", &encryptedPDU[7], CTlen -1);
        blockprint_payload("CRC =            ", &encryptedPDU[lengthOfCTPDU -4], 4);
        printf("\n\n");
#endif

        /* 3. Decrypt Cipher Text to decrypt text */
        decrypt_pdu(key,&encryptedPDU[6],lengthOfPlainText+1,decrypt);

#ifdef DEBUG1
        printf("DECRYPT TEXT: %d Byte\n\n",lengthOfPlainText);
        blockprint_payload("decrypted text",decrypt,lengthOfPlainText);
        printf("\n\n");
#endif

        /* 4. Compare decrypt text and original plain text */
        if ( compare(decrypt,&plainPDU[6],lengthOfPlainText) == 0 ) {
                return 1; /* Test Success */
        } else {
                return 0; /* Test Failure */
        }
}

/***************************************************/
/* AES CTR main(int argc, char* argv[])            */
/* Test vectors                                    */
/***************************************************/
int main(int argc, char* argv[])
{
        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;
}
```

[delete annex F]