Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**
Title	**MBS AES-CTR Test Vector**
Date Submitted	**2004-08-17**
Source(s)	JUNHYUK SONG, YONG CHANG, JICHEOL LEE Samsung Electronics — Voice: +82-31-279-3639 junhyuk.song@samsung.com Voice: +82-31-279-3639 jicheol.lee@samsung.com
Re:	IEEE P802.16e/D4-2004
Abstract	Proposal for MBS AES-CTR Test Vector
Purpose	Review and Adopt the suggested changes into P802.16e/D4
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>.

MBS AES-CTR Test Vector

JUNHYUK SONG, YONG CHANG,JICHEOL LEE
Samsung Electronics

Introduction

Per discussion in last Portland meeting, 802.16e D4 supports AES CTR mode for MBS security. In this contribution, we propose Known Answer Test (KAT) test vectors and example program for AES CTR mode. KAT Test Vectors and example program shall be used to determine the correctness of implementations of 802.16e cryptographic method from various vendors, and shall improve inter-operability to interface between 802.16e compliant systems. This contribution propose following known answer test routine:

1. Input known Test vectors
2. Computation Known Test Vectors according to 802.16e, FIPS 197 and NIST Special Publication 800-38A (AES-CTR)
3. Comparison of the computed result against the known answer
4. Verify whether computed results are equal to the known value

Known Answer Test for Variable Text

In this variable text KAT, we propose three variable size test vectors, and test program for AES CTR mode. The 16byte fixed size key and 32bits nonce and 64/256/1500 bytes plain texts randomly generated by GNU gcc rand() are given for the test. The AES encryption and decryption function defined in the test program has passed the 128bits Key size, Known Answer Test (KAT) and Monte Carlo TEST (MCT) required by NIST [1]. It will be assumed that correctly developed S/W or H/W implementation of AES-CTR shall be able to produce 802.16 MPDU that contains 32bits nonce and encrypted PDU with proper handing of the remainder according to 802.16e and NIST Special Publication 800-38A. The correctly developed S/W or H/W implementation of AES-CTR should have the same result in this KAT if the same test vectors are given as input.

Proposed baseline text

7.8.2.3 Cryptographic Method Test Vectors

7.8.2.3.1 AES OMAC Mode Known Answer Test for Variable Text

7.8.2.3.1.1 TEST Vector

TBD

7.8.2.3.1.1 TEST Program

TBD

7.8.2.3.2 AES CCM Mode Known Answer Test for Variable Text

7.8.2.3.2.1 TEST Vector

TBD

7.8.2.3.2.1 TEST Program

TBD

7.8.2.3.3 AES CTR Mode Known Answer Test for Variable Text

7.8.2.3.3.1 TEST Vector

Test 1:

PLAIN 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
```

Nonce: 4 Byte

```
22 22 1a 70
```

Counter: 16 Byte

```
22 22 1a 70 22 22 1a 70 22 22 1a 70 22 22 1a 70
```

KEY: 16 Byte

```
00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff
```

CIPHER TEXT: 64 Byte + 4Byte Nonce

```
22 22 1a 70 b6 72 f2 af 6a cc 20 ae ee 1a d8 14
12 8c 31 8b 95 5b be 80 5b 38 92 49 89 76 00 f5
20 74 54 32 7d 6d 0f b4 ac 0a 94 f3 7c a0 9e 45
05 33 98 fe a8 9c 20 0a d3 58 12 6d 9e 89 a4 05
26 5c 96 e7
```

DECIPHERED 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
```

Test 2:

PLAIN TEXT: 256Byte

```
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 d7
```

Nonce: 4 Byte

```
5c b4 4a 05
```

Counter: 16 Byte

```
5c b4 4a 05 5c b4 4a 05 5c b4 4a 05 5c b4 4a 05
```

KEY: 16 Byte

```
00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff
```

CIPHER TEXT: 256 Byte + 4Byte Nonce

```
5c b4 4a 05 8b 34 7e 83 50 f9 73 01 1c 93 34 8b
51 b4 43 87 b5 6b b8 72 b3 45 78 bd c6 1f fb 46
16 98 f8 0b cd cd b3 d2 2a b1 17 c3 9d f5 49 58
65 9e b5 7e 56 7a b6 4a f9 46 0e 6a 33 04 fa a8
a1 a2 01 4c cd b3 d8 7c 49 91 1b 6b d5 9c 87 b4
6d bd ee 8d 36 0c 4f f7 67 38 6e 2a eb 7c 08 54
4e 12 16 74 39 db 14 38 71 f5 54 49 04 f6 0e 4a
cc 77 30 ee ff a9 97 bf f2 23 ba 2c c7 da aa 5a
0d 05 9d 0c 5a ee 9d d8 70 f2 df d1 79 c1 a2 6d
65 fc bb 59 ad f2 3d 7f 8f 4c a8 f4 ce f5 98 bf
1f c4 5c b7 e8 82 d6 5a 28 77 8d 21 b0 97 94 e8
92 c4 a5 2a 78 fe cd 0b 5c a0 35 5b 7a 44 4a c4
04 be bb 34 b6 cb 74 e4 14 08 08 d8 0b 87 6b 10
fa 08 4a 6c 77 8b 6b a1 00 9e 3f 1b b0 e7 6f fa
06 6b 2d 47 f4 7e ab cf 69 14 3b f9 97 92 95 44
42 ee 00 8e 68 9c 0f 96 c4 75 38 cc 6a 0f 1d af
d6 24 57 1c
```

DECIPHERED TEXT: 256 Byte

```
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 d7
```

Test 3:

PLAIN TEXT: 1500 Byte

```
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 c4 b1
```

Nonce: 4 Byte

```
18 26 e4 11
```

Counter: 16 Byte

```
18 26 e4 11 18 26 e4 11 18 26 e4 11 18 26 e4 11
```

KEY: 16 Byte

```
00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff
```

CIPHERED TEXT: 1500 Byte

```
18 26 e4 11 99 7d 26 f2 bf 10 dc f3 4c b4 e3 c3
63 79 d0 5f 52 99 fb fc 3d bf a9 9c 11 5e b0 2a
de f1 e4 c6 53 97 4a b5 8e 4f f8 81 d4 0e 0b e6
79 e1 70 5b c0 96 82 22 41 04 2a c8 67 e3 8d 18
65 27 8f 52 4b cd 05 0e c3 92 59 dd 4d d9 e3 ef
a1 90 c1 66 ee ce 41 95 84 b5 37 1a 54 ed c9 e3
63 c9 ac 67 59 2f f3 0a f7 70 06 d3 8d 36 d4 07
00 e8 fd dd 1d 0e cf d8 d5 60 9f de b8 91 5d ed
f9 7c 06 61 e9 59 50 4e 1e aa 53 89 32 70 c7 74
6e 41 b1 d0 5e e8 78 5d d8 ad a3 b9 da 94 98 1c
6d 7c 61 30 71 e0 fa 60 9b ff 45 71 54 8b 9c 60
11 de 5d b9 cc a0 e8 35 79 3e 0a 9f d4 08 9f f9
1b 0a 21 a8 4a 39 90 c4 07 c4 80 1b 2c 56 ca fb
3e ad a6 0c f3 49 8f f3 96 b9 73 a4 e0 ef bb 34
b5 1a b9 69 b4 81 61 1a c0 16 64 8d 10 5e f2 ff
28 36 35 e6 75 20 a5 00 97 23 d8 f0 96 9a df 1b
ec b2 1d 8b 77 d6 44 4d 72 28 c4 48 b6 46 ad 87
c0 3a 9f 76 79 f4 f0 a8 33 08 23 27 3d 3c 27 82
98 33 3d c7 bf 12 b9 8a 18 76 76 99 50 8e 61 4a
9e a6 60 88 18 94 63 74 8f ff 14 ec f5 a5 ff 38
13 84 ff a4 76 2d 1c 8c 45 ad 98 14 d5 49 fd 32
f9 42 f3 75 1e 57 c1 54 43 72 e2 fe 91 6e e5 36
3e 08 20 9f a4 4f 61 a3 e0 a1 fd 02 5d 0c 47 ec
8c ac 76 ef 8b ed 60 e5 8f 41 8f 11 6a 42 87 5c
43 c5 8f e3 89 36 c7 91 49 3f df 46 17 3e ec d2
9a 80 c2 eb 63 ab e3 63 33 73 bd fd e1 8d 53 31
79 b7 e8 c4 5f 1b 4b 87 88 33 76 d5 3c bb cf bf
d7 27 4b bc 9f 05 a2 32 69 8f 95 55 90 18 d9 a8
65 fc 03 3e 44 63 7f 21 6b ca c6 7f 96 f1 d2 c5
ac 0c 96 25 d9 63 15 e6 0d d6 5b d7 6d 8e 37 77
a7 27 c7 bb e8 a0 17 3e 1e 36 a5 7c e5 7e 91 62
85 d5 cf 97 20 32 2b a7 72 f1 d5 54 a8 b5 ea d4
8d 3d 76 0a 2f 92 30 d8 3e d0 f3 52 35 f3 bc 8b
06 a7 13 41 c4 e8 51 e5 53 a2 e5 05 28 d8 92 96
a2 c5 ba 87 f0 20 f7 25 6c b9 c7 5c 21 32 1c 50
42 ba 05 cb d1 fa e5 7b 18 22 ec c7 be 84 27 62
e2 95 3a ad c4 34 63 8b d0 bf 4c 64 27 50 d9 22
80 85 bd 4b be ac 89 81 a4 5c 4c 86 75 b9 84 a2
ff 92 5f 9b 56 1d 57 b3 7b 0f e2 36 95 c4 55 f3
a2 ed 08 9a ab df 2e 9c bc d0 54 3b b6 d3 3c 9e
a4 44 e2 3d 8c c9 46 e5 89 42 6e ed 35 f6 a3 37
60 99 ce 55 e0 51 31 f4 0a d7 99 91 ce a7 94 23
bf e3 a1 20 f5 bb 7e f7 39 a6 67 32 6b 43 40 3a
cb d6 62 1d 99 b7 c6 ab c0 45 0f c4 56 00 f4 37
43 05 73 f2 74 ee 27 bf 86 dd 72 f6 43 27 c9 5f
7d 6c 10 8d c7 fa 78 5a 81 bc de d1 34 7f 29 a9
0c 54 cd 17 96 01 75 ef ec 90 ca 0b 13 dd 93 95
16 22 d4 80 47 4f 53 15 e4 7c cd ba 67 08 af 3d
56 55 2e e8 c6 70 e7 fe 4d e6 da c8 f2 2c 15 1d
eb 41 46 74 db 11 28 42 6e f3 42 de 00 ec ba 97
7d b3 d6 42 16 8e 48 11 f6 bd 30 25 b7 55 c7 98
67 7d de a3 be f4 bd 72 8b 5d 94 e4 9f 92 43 e7
97 c7 f1 e6 be b9 10 d2 ee 36 9c 4a ae 98 fa ab
```

```
6d 8e 53 48 fe 15 84 11 0f 27 09 d0 ca c2 60 2d
c1 22 29 cf 95 aa bb 2f 94 0d 68 b0 74 6c 11 15
b4 79 48 ce 44 e0 59 6e 0f 40 5d d8 e3 9f 3b 24
b6 10 13 47 da b5 53 ae cc b7 70 92 73 30 65 a8
34 66 67 56 66 77 28 1a 31 13 52 42 7c 52 f5 0e
a7 a8 2e b9 ee 9f c3 c5 21 96 81 3e 71 af 06 44
fc be a9 5c 4b c0 21 0e 20 ff fd 5a 7c 2c e9 ba
ba cd f8 af b7 71 b4 18 b7 9c 75 d7 bc 76 b8 6c
c7 97 2c 74 82 18 19 71 07 65 5f e0 6e 86 34 66
28 50 6d 99 2d 76 94 01 e0 3b cc 9e 4d 77 72 fe
14 ce 9c 05 6f a5 05 bc 14 b5 59 32 b9 3e 26 92
7c 05 4f a1 4d 05 14 c7 7e 83 8a 3a 89 42 af ae
56 56 da 60 62 c4 6e 6f 24 87 fc 58 67 56 46 d5
40 a0 cb 83 9d e6 a4 d4 cb e1 f1 9d 66 36 37 fb
56 c1 00 61 ec 86 15 aa 6f 7e af 28 f5 e7 19 f7
88 13 77 2d ad cf 74 a8 db 77 0e 57 f0 8c 11 e7
4b e5 ef 9d a4 d7 df 54 d2 a0 75 27 9f b4 01 dc
94 d9 4e bd f7 8a e6 0d ea a3 0d f7 a3 55 2e bc
1b 94 c3 66 06 13 2e 11 99 31 c0 4f a4 9f 70 ca
88 69 87 92 0e 06 22 32 c4 d7 4d 86 25 d5 61 71
f9 02 f7 fb 90 fa ff 02 e9 21 52 5b 5d bc 87 af
c9 2b 65 66 62 0c 41 fb 9b 0f 67 00 28 5c c2 f8
33 11 c5 9e e5 a3 08 96 30 30 7e 2d f0 c8 c5 5e
c8 cf 5c 5c 4b f6 07 6a 04 31 04 37 27 63 6a c7
0c 31 7f 9a 75 15 0d f3 14 75 16 c6 90 92 90 f0
4c 0f 2c 3f a5 c7 01 c3 a1 9a 2f 0a 81 99 90 ae
8b f6 08 b2 ab f4 cd a5 ca fb c3 94 d5 85 09 d8
0b 4e 96 88 bf 33 ed 28 52 11 b9 74 47 ed 7b 55
23 ad 5e 1a 41 aa 16 08 00 42 08 2b de e8 c8 2c
14 e1 6a ab 34 c7 2d f6 82 fc b2 69 0d 9e a5 6c
84 f2 cd 6b ae 28 9a ab 59 34 33 a4 b2 61 20 02
d1 74 eb d8 0c 3a 72 8f 72 b8 68 24 c9 15 91 1a
3a e0 6d 33 f5 1b ee 86 e3 1c 6a 42 96 af b4 f5
f3 ef 55 98 10 10 cf aa ac 95 71 3c dd 74 2e be
47 f8 d6 14 3b b1 d1 6f 83 ab 57 bf ff 36 40 42
6f 6c 82 46 4e d8 05 81 ac 6f aa 4f 3c fe a5 3f
b3 fd 13 9e bb 91 a0 f8 7a c1 95 e8 9a 28 b7 9a
64 26 e5 e7 fb ce 60 09 4f 3e e3 79 8a df 14 b8
17 eb 14 7d ff d3 ba a1 a7 c2 60 c8 5e 5f 34 0d
5e 7b d2 02 d4 3f 87 47 ca b6 54 02 eb f4 69 20
49 54 50 1c 01 d6 09 1d c5 5d 78 c5 38 af 53 72
```

DECIPHERED TEXT: 1500Byte

```
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 c4 b1
```

7.8.2.3.3.2 TEST Program

```
/******************************************************************/
/* 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: 0.1                                               */
/* 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 */
/*        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;
            }
}

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

```
/* It initialize 128bit counter by concaternating   */
/* the same 32 bit nonce four times                 */
/**************************************************/

void init_counter(unsigned char *nonce32, unsigned char *ctr)
{
        int             i, j;

        for ( i=0; i<4; i++ ) {
                for ( j=0; j<4; j++ ) {
                        ctr[i*4+j] = nonce32[j];
                }
        }
}

/*************************************************************/
/* 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;
        }
}
void generate_nonce(unsigned char *nonce)
{
        unsigned long   value = htonl(random_32bit());

        memcpy(nonce,(char*)&value,4);
}

/*******************************************************/
/* int encrypt_pdu()                                   */
/* Encrypts a plaintext pdu in accordance with         */
/* the proposed 802.16e AES CTR specification.         */
/* Nonce insertion 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],nonce[4];
```

```
            unsigned char        aes_out[16], remain[16], temp[16];

            generate_nonce(nonce);
#ifdef DEBUG
            printf("Generate 32bit nonce : ");
            print_hex(nonce,4);
#endif

            for (i=0; i<4; i++)
                        cipher[i] = nonce[i];

            out_len += 4;

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

            init_counter(nonce,ctr);
#ifdef DEBUG
            printf("Initialize Counter: ");
            print_hex(ctr,16);
     printf("Key: ");
     print_hex(key,16);
#endif
            for ( i=0; i< n_blocks; i++ ) {
                        aes128k128d(key, ctr, aes_out);
                        bitwise_xor(aes_out, &plain[i*16], &cipher[i*16+4]);
                        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+4+i] = temp[i];
            }
            out_len += n_remain;
            return out_len;
}
/*****************************************************/
/* int decrypt_pdu()                                          */
/* decrypts a plaintext pdu in accordance with              */
/* the proposed 802.16e AES CTR specification.               */
/* Nonce insertion takes place.                              */
/* Returns the resulting cipher 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],nonce[4];
        unsigned char       aes_out[16], remain[16], temp[16];

        for ( i=0; i<4; i++ ) {
                nonce[i] = cipher[i];
        }

        len -= 4;

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

        init_counter(nonce,ctr);
        for ( i=0; i< n_blocks; i++ ) {
             aes128k128d(key, ctr, aes_out);
             bitwise_xor(aes_out, &cipher[i*16+4], &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+4+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 --------------------------------------₩n");
            print_hex(plain,length);
#endif

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

            encrypt_pdu(key,plain,length,cipher);

#ifdef DEBUG
            printf("CIPHER TEXT --------------------------------------₩n");
            print_hex(cipher,length+4);
#endif

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

            decrypt_pdu(key,cipher,length+4,decrypt);

#ifdef DEBUG
            printf("DECRYPT TEXT --------------------------------------₩n");
            print_hex(decrypt,length);
#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()                                 */
/* Iterate through the test cases, passing them   */
/* through the ccm algorithm to produce test      */
/* vectors                                        */
/**************************************************/

int main()
{
        int                     i, len[] = { 64, 256, 1500, 10000 };

        for ( i=0; i<sizeof(len)/sizeof(len[0]); i++ ) {
                printf("Test Case with length = %04d₩n",len[i]);
                if ( test_case(len[i]) ) {
                        printf(" ==> Success₩n");
                } else {
                        printf(" ==> Failure₩n");
                }
        }
        return 0;
}
```