Description of Known Answer Tests and Monte Carlo Tests
for Advanced Encryption Standard (AES) Candidate Algorithm Submissions

Original: January 7, 1998
Update: January 13, 1998 *(updated by adding Sections 3.3 and 3.4; modified chart in Sec. 5)*
Update: February 17, 1998 *(corrections in Sec.2.2, 2nd para., 2nd sentence, and Sec. 4.1, 2.b.iii to refer to 128-bit blocks)*

Table of Contents

* * *

1. Overview

Each submission package is required to include Known Answer Test (KAT) and Monte Carlo Test (MCT) values, which can be used to determine the correctness of an implementation of the candidate algorithm. Values shall be included (at a minimum) for each of the three minimum required key sizes: 128, 192, and 256 bits.

These KAT and MCT tests are based on tests specified in the draft NIST Special Publication 800-17, *Modes of Operation Validation System (MOVS): Requirements and Procedures* [MOVS], which describes tests for the DES and Skipjack algorithms (two examples of block cipher algorithms). Each

of the tests for which values are required in the submission packages is described below. In addition, example files are included which specify the exact syntax and format which submitters are required to use when submitting their KAT and MCT values.

2. Modes of Operation - Description

KAT values are required for operation of the AES candidate algorithms in the Electronic Codebook (ECB) mode only (for all three required key sizes). MCT values are required for operation of the AES candidate algorithms in the ECB *and* Cipher Block Chaining (CBC) modes of operation. These modes are described briefly in the following two sections, which are based on information in [MOVS].

2.1 Electronic Codebook (ECB) Mode

The Electronic Codebook (ECB) Mode is diagramed in Figure 1. In ECB encryption, a plaintext data block (D1, D2, . . ., D128) is used directly as the input block (I1, I2, . . ., I128). The input block is then processed through the algorithm in the encrypt state. The resulting output block (O1, O2, . . ., O128) is used directly as ciphertext (C1, C2, . . ., C128).

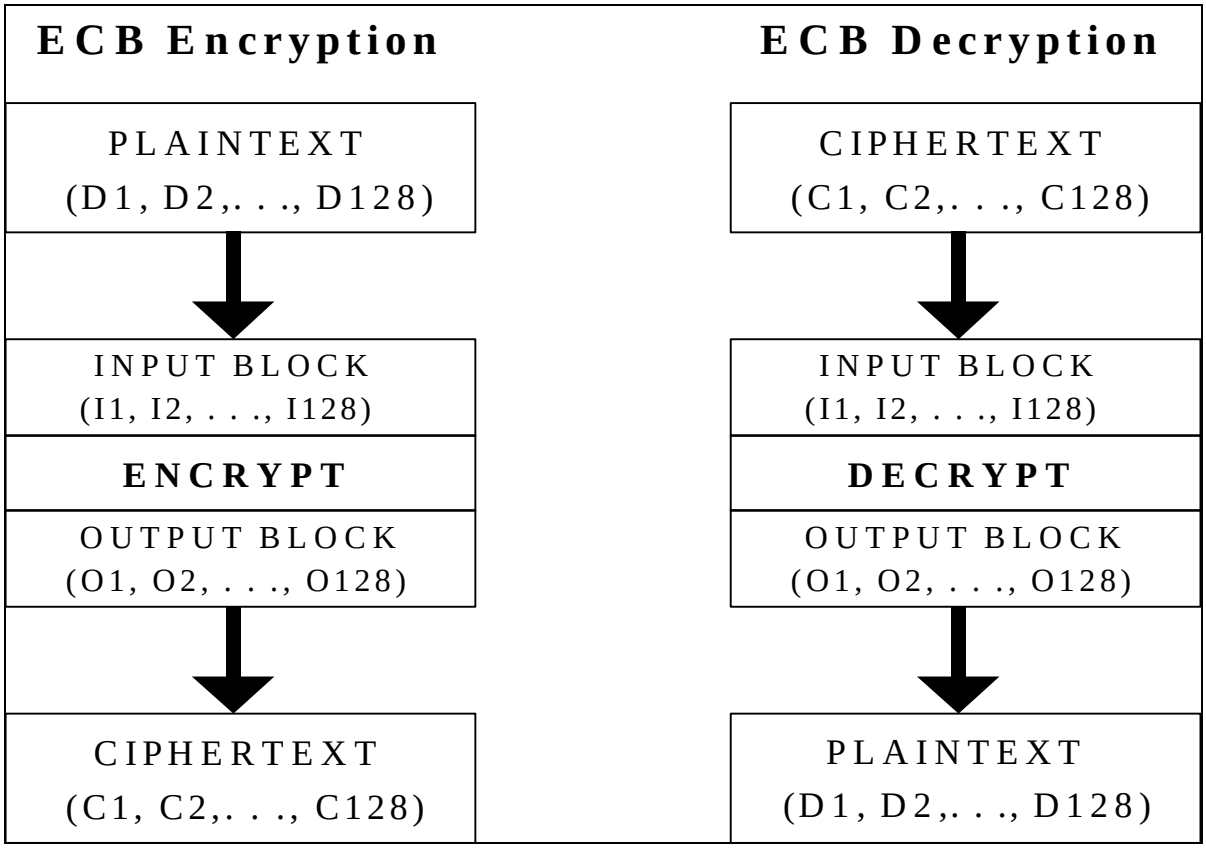
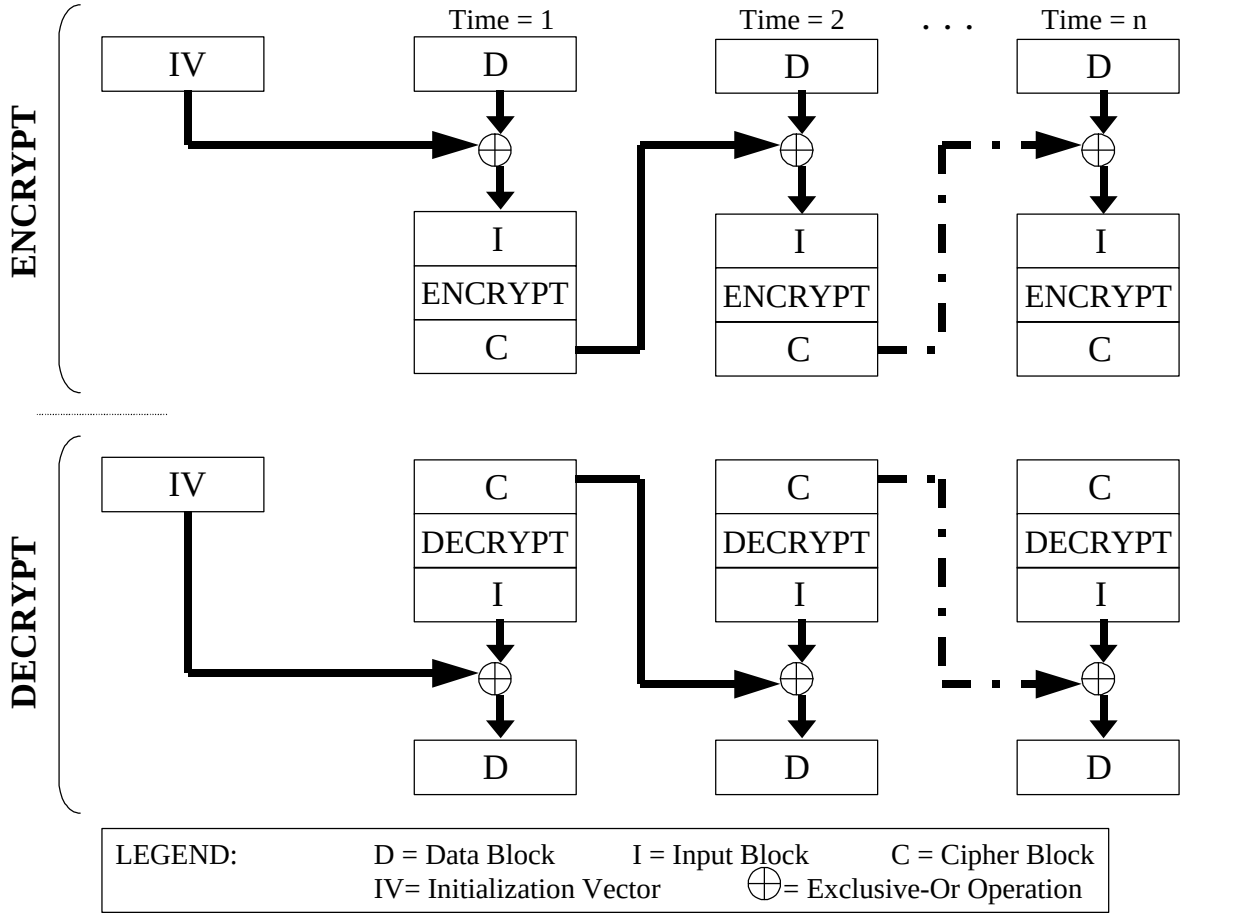


Figure 1: *Electronic Codebook (ECB) Mode*

Cipher Block Chaining (CBC) *(NOTE: All variables are 128 bits in length.)*



processed through the algorithm in the encrypt state, and the resulting output block is used as the ciphertext, i.e., (C1, C2, . . ., C128) = (O1, O2, . . ., O128). This first ciphertext block is then exclusive-ORed with the second plaintext data block to produce the second input block, i.e., (I1, I2, . . ., I128) = (C1⊕D1, C2⊕D2, . . ., C128⊕D128). Note that I and D now refer to the second block. The second input block is processed through the algorithm in the encrypt state to produce the second ciphertext block. This encryption process continues to "chain" successive cipher and plaintext blocks together until the last plaintext block in the message is encrypted.

In CBC decryption, the first ciphertext block of an encrypted message is used as the input block and is processed through the algorithm in the decrypt state, i.e., (I1, I2, . . ., I128) = (C1, C2, . . ., C128). The resulting output block, which equals the original input block to the algorithm during encryption, is exclusive-ORed with the IV (which must be the same as that used during encryption) to produce the first plaintext block, i.e., (D1, D2, . . ., D128) = (O1⊕IV1, O2⊕IV2, . . ., O128⊕IV128). The second ciphertext block is then used as the next input block and is processed through the algorithm in the decrypt state. The resulting output block is exclusive-ORed with the first ciphertext block to produce the second plaintext data block, i.e., (D1, D2, . . ., D128) = (O1⊕C1, O2⊕C2, . . ., O128⊕C128). Note that again the D and O refer to the second block. The CBC decryption process continues in this manner until the last complete ciphertext block has been decrypted. Ciphertext representing a partial data block must be decrypted in a manner as specified for the application.

The above processes for encryption and decryption in CBC mode are independent of key size.

3. Known Answer Tests (KATs)

Known Answer Test values must be provided with submissions, which demonstrate operation of the AES candidate algorithm in ECB mode, for each of the minimum required key values (128, 192, and 256 bits). There are two types of KATs that are required for all submissions: 1) Variable Key KAT, and 2) Variable Text KAT. Section 3.3 describes a Tables KAT, which is required only for those candidate algorithms which use tables.

3.1 Variable Key Known Answer Tests

The ECB mode of the algorithm shall be used for the variable key KAT, and the submitter shall use the submitted algorithm to generate these test values as follows. The 128-bit plaintext shall always be initialized to zero, for each encryption. Each key used to encrypt the zero plaintext block is represented as a basis vector, consisting of a "1" in the i^{th} position and "0" in all of the other positions. The input block is processed through the algorithm in the encrypt state to produce ciphertext. Each of the possible key basis vectors is tested in this manner, by shifting the “1" a single position at a time, starting at the most significant (left-most) bit position of the key. This shall result in 128, 192, or 256 plaintext-ciphertext pairs - this number corresponds to the size of the key being tested. The resulting 128, 192, or 256, index-key-ciphertext triples shall be recorded by the submitter in the file “`ecb_vk.txt`”.

The above shall be repeated for each of the three minimum key sizes (128, 192, and 256 bits).

See the example file "`ecb_vk.txt`" for the required formatting and syntax of submitted test values.

To test the algorithm's decrypt state, the ciphertext shall be input into the algorithm (in the decrypt state) with the corresponding key value (basis vector), and the result must equal the zero plaintext value.

3.2 Variable Text (Plaintext/Ciphertext) Known Answer Tests

The ECB mode of the algorithm shall be used for the variable text KAT, and the submitter shall use the submitted algorithm to generate these test values as follows. The key shall be initialized to zero. Each block of data input into the algorithm is represented as a 128-bit basis vector, consisting of a "1" in the i^{th} position and "0" in all of the other positions. The input block is processed through the algorithm in the encrypt state to produce ciphertext. Each of the basis vectors is tested in this manner, by shifting the "1" a single position at a time, starting at the most significant (left-most) bit position. This shall result in 128 plaintext-ciphertext pairs. The 128 index-plaintext-ciphertext triples shall be recorded by the submitter in the file "`ecb_vt.txt`".

The above shall be repeated for each of the three minimum key sizes (128, 192, and 256 bits).

See the example file "`ecb_vt.txt`" for the required formatting and syntax of submitted test values.

To test the algorithm's decrypt state, the ciphertext shall be input into the algorithm (in the decrypt state) with the key initialized to zero, and the result must equal the corresponding plaintext value.

3.3 Tables Known Answer Tests *(if applicable)*

If tables are used as part of the candidate algorithm, then the submitter shall include KAT values that collectively exercise every table entry (when operating the algorithm in the ECB mode). The values shall consist of *n* key-plaintext-ciphertext triples, where the key and plaintext values are determined by the submitter. These *n* triples test all tables by forcing every entry in all the tables to be used at least once. The index-key-plaintext-ciphertext sets shall be recorded by the submitter in the file "`ecb_tbl.txt`". (E.g., There are 19 specific key-plaintext-ciphertext triples that can be used to [collectively] test every entry in all eight of the Substitution Tables in DES.)

The above shall be repeated for each of the three minimum key sizes (128, 192, and 256 bits).

See the example file "`ecb_tbl.txt`" for the required formatting and syntax of submitted test values. As indicated in the example file, the submitter shall also include a brief description of what tables are being tested.

To test the algorithm's decrypt state, the ciphertext shall be input into the algorithm (in the decrypt state) with the corresponding key value, and the result must equal the corresponding plaintext value.

For those candidate algorithms which do not use tables, the tests and requirements specified in this section do not apply.

3.4 Intermediate Values Known Answer Tests *(if applicable)*

In Section 2.B.3 of the Request for Candidate Algorithm Nominations for the AES, part a) iii. specifies that additional known answer tests shall be included "if the candidate algorithm calculates intermediate values (e.g., internal rounds) for an encryption or decryption operation".

The file(s) containing values for the Intermediate Values KAT shall have filenames which are appropriate, and shall contain a description of what is being tested. The file(s) shall also have a format that is as similar to that of the example KAT files as possible; the exact information included in the file(s) will likely depend on the algorithm, and therefore no example file is given for this KAT.

For those candidate algorithms which do not calculate intermediate values, the tests and requirements specified in this section do not apply.

4. Monte Carlo Tests (MCTs)

Monte Carlo Test values must be provided with submissions, which demonstrate operation of the AES candidate algorithm in *both* ECB and CBC modes, for *each* of the minimum required key values (128, 192, and 256 bits), for both the encrypt and decrypt states. There are two types of MCTs, for the encrypt and decrypt states. Both are described in the following sections.

Note that the initial plaintext and IV values used in the submitted file shall be determined by the submitter. The values contained in the example file are only meant to demonstrate the proper syntax and format for the submission - they are NOT required for use by the submitter in generating values.

Each Monte Carlo Test consists of four million cycles through the candidate algorithm implementation. These cycles are divided into four hundred groups of 10,000 iterations each. Each iteration consists of processing an input block through the candidate algorithm, resulting in an output block. At the $10{,}000^{th}$ cycle in an iteration, new values are assigned to the variables needed for the next iteration. The results of each $10{,}000^{th}$ encryption or decryption cycle are recorded and included by the submitter in the appropriate file.

```
Initialize KEY0, PT0

FOR i = 0 TO 399
  {
        Record i, KEYi, PT0
        FOR j = 0 TO 9,999
         {
              IBj = PTj
              Perform algorithm in encrypt state, resulting in CTj
              PTj+1 = CTj
         }
        Record CTj

        KEYi+1= KEYi ⊕ last n bits of CT, where n=128, 192, or 256 (depending on key size)
        PT0 = CT9999
  }
```

Figure 3:

CT (CT_{9998}) with the 128 bits of the current CT (CT_{9999}). This value shall then be exclusive-ORed with the current KEY to form the new KEY; e.g., if the size of KEY is 192 bits, ($KEY1_{i+1}$, $KEY2_{i+1}$, . . ., $KEY192_{i+1}$) = ($KEY1_i \oplus CT65_{9998}$, $KEY2_i \oplus CT66_{9998}$, . . ., $KEY64_i \oplus CT128_{9998}$, $KEY65_i \oplus CT1_{9999}$, $KEY66_i \oplus CT2_{9999}$, ... $KEY192_i \oplus CT128_{9999}$).

e. Assign a new value to PT in preparation of the next outer loop. PT_0 shall be assigned the value of the current CT, i.e., ($PT1_0$, $PT2_0$, . . .,$PT128_0$) = ($CT1_{9999}$, $CT2_{9999}$, . . .,$CT128_{9999}$). (Note that the new PT shall be denoted as PT_0 to be used for the first pass through the inner loop when j=0.)

NOTE: The recorded output for this test shall consist of 400 sets of (i, KEY, PT, CT).

4.2 ECB Decrypt MCT

```
Initialize KEY0, CT0

FOR i = 0 TO 399
 {
        Record i, KEYi, CT0
        FOR j = 0 TO 9,999
         {
                IBj = CTj
                Perform algorithm in decrypt state, resulting in PTj
                CTj+1 = PTj
         }
        Record PTj

        KEYi+1 = KEYi ⊕ last n bits of PT,  where n=128, 192, or 256 (depending on key size)
        CT0 = PT9999
 }
```

Figure 4: *Monte Carlo Test - ECB Decryption*

As summarized in Figure 4, the Monte Carlo Test for the ECB Decrypt state shall be performed as follows:

1. The submitter shall initialize KEY and ciphertext (CT) variables. The CT shall consist of 128 bits, while the KEY length shall be either 128, 192, or 256 bits.

2. The submitter shall perform the following for i=0 through 399:
   a. Record the current values of the outer loop number i, KEY_i, and CT_0.
   b. Perform the following for j=0 through 9999:

i. Set the input block IB_j equal to the value of CT_j, i.e., $(IB1_j, IB2_j, \ldots, IB128_j) = (CT1_j, CT2_j, \ldots, CT128_j)$.

ii. Process IB_j through the candidate algorithm in the decrypt state, resulting in plaintext PT_j.

iii. Prepare for loop j+1 by assigning CT_{j+1} with the current value of PT_j, i.e., $(CT1_{j+1}, CT2_{j+1}, \ldots, CT128_{j+1}) = (PT1_j, PT2_j, \ldots, PT128_j)$.

c. Record PT_j .

d. Assign a new value to the KEY in preparation for the next outer loop. The new KEY shall be calculated by exclusive-ORing the current KEY with the current PT. For example if the size of KEY is 128 bits, this equates to $(KEY1_{i+1}, KEY2_{i+1}, \ldots, KEY128_{i+1}) = (KEY1_i \oplus PT1_{9999}, KEY2_i \oplus PT2_{9999}, \ldots, KEY128_i \oplus PT128_{9999})$.

For cases when the size of KEY is 192 or 256 bits, PT shall be expanded in length to 192 or 256 bits (as appropriate) before the new KEY can be formed. This expansion shall be accomplished by concatenating the 64 or 128 rightmost bits of the previous PT (PT_{9998}) with the 128 bits of the current PT (PT_{9999}). This value shall then be exclusive-ORed with the current KEY to form the new KEY; e.g., if the size of KEY is 192 bits, $(KEY1_{i+1}, KEY2_{i+1}, \ldots, KEY192_{i+1}) = (KEY1_i \oplus PT65_{9998}, KEY2_i \oplus PT66_{9998}, \ldots, KEY64_i \oplus PT128_{9998}, KEY65_i \oplus PT1_{9999}, KEY66_i \oplus PT2_{9999}, \ldots KEY192_i \oplus PT128_{9999})$.

e. Assign a new value to CT in preparation of the next outer loop. CT_0 shall be assigned the value of the current PT, i.e., $(CT1_0, CT2_0, \ldots, CT128_0) = (PT1_{9999}, PT2_{9999}, \ldots, PT128_{9999})$. (Note that the new CT shall be denoted as CT_0 to be used for the first pass through the inner loop when j=0.)

NOTE: The recorded output for this test shall consist of 400 sets of (i, KEY, CT, PT).

4.3 CBC Encrypt MCT

```
Initialize KEY0, IV, PT0

FOR i = 0 TO 399
 {
        If (i==0) CV0 = IV
        Record i, KEYi, CV0, PT0
        FOR j = 0 TO 9,999
         {
                IBj = PTj ⊕ CVj
                Perform algorithm in encrypt state, resulting in CTj
                IF j=0
                  PTj+1 = CV0
                ELSE
                  PTj+1 = CTj-1
                CVj+1 = CTj
         }
        Record CTj

        KEYi+1 = KEYi ⊕ last n bits of CT, where n=128, 192, or 256 (depending on key size)
        PT0 = CT9998
        CV0 = CT9999
 }
```

Figure 5: *Monte Carlo Test - CBC Encryption*

As summarized in Figure 5, the Monte Carlo Test for the CBC Encrypt state shall be performed as follows:

1. The submitter shall initialize the KEY, initialization vector (IV) and plaintext (PT) variables. The PT and IV shall consist of 128 bits each, while the KEY length shall be either 128, 192, or 256 bits..

2. The submitter shall perform the following for i = 0 through 399:
   a. If i=0 (if this is the first time through this loop), set the chaining value CV_0 equal to the IV.
   b. Record the current values of the outer loop number i, KEY_i, CV_0 and PT_0.
   c. Perform the following for j = 0 through 9999:
      i. Set the input block IB_j equal to the value of PT_j exclusive-ORed with the CV_j, i.e., $(IB1_j, IB2_j, \ldots, IB128_j) = (PT1_j \oplus CV1_j, PT2_j \oplus CV2_j, \ldots, PT128_j \oplus CV128_j)$.
      ii. Process IB_j through the candidate algorithm in the encrypt state, resulting in CT_j.
      iii. Prepare for loop j+1 by doing the following:

- Assign CV_{j+1} with the current value of CT_j, i.e., ($CV1_{j+1}$, $CV2_{j+1}$, . . ., $CV128_{j+1}$) = ($CT1_j$, $CT2_j$, . . ., $CT128_j$).
- If the inner loop being processed is the first loop, i.e., j = 0, assign PT_{j+1} with the current value of CV_0, i.e., ($PT1_1$, $PT2_1$, . . ., $PT128_1$) = ($CV1_0$, $CV2_0$, . . ., $CV128_0$). Otherwise, assign PT_{j+1} with the CT from the previous inner cycle, CT_{j-1}, i.e., ($PT1_{j+1}$, $PT2_{j+1}$, . . .,$PT128_{j+1}$) = ($CT1_{j-1}$, $CT2_{j-1}$, . . ., $CT128_{j-1}$).

d. Record CT_j .

e. Assign a new value to KEY in preparation for the next outer loop. The new KEY shall be calculated by exclusive-ORing the current KEY with the current CT. For example if the size of KEY is 128 bits, this equates to ($KEY1_{i+1}$, $KEY2_{i+1}$, . . ., $KEY128_{i+1}$) = ($KEY1_i \oplus CT1_{9999}$, $KEY2_i \oplus CT2_{9999}$, . . ., $KEY128_i \oplus CT128_{9999}$).

For cases when the size of KEY is 192 or 256 bits, CT shall be expanded in length to 192 or 256 bits (as appropriate) before the new KEY can be formed. This expansion shall be accomplished by concatenating the 64 or 128 rightmost bits of the previous CT (CT_{9998}) with the 128 bits of the current CT (CT_{9999}). This value shall then be exclusive-ORed with the current KEY to form the new KEY; e.g. if the size of KEY is 192 bits, ($KEY1_{i+1}$, $KEY2_{i+1}$, . . ., $KEY192_{i+1}$) = ($KEY1_i \oplus CT65_{9998}$, $KEY2_i \oplus CT66_{9998}$, . . ., $KEY64_i \oplus CT128_{9998}$, $KEY65_i \oplus CT1_{9999}$, $KEY66_i \oplus CT2_{9999}$, . . ., $KEY192_i \oplus CT128_{9999}$).

f. Assign a new value to CV_0 in preparation for the next outer loop. CV_0 shall be assigned the value of the current CT, i.e., ($CV1_0$, $CV2_0$, . . ., $CV128_0$) = ($CT1_{9999}$, $CT2_{9999}$, . . ., $CT128_{9999}$). (Note that the new CV shall be denoted as CV_0 because this value is used for the first pass through the inner loop when j=0.)

g. Assign a new value to the PT in preparation of the next outer loop. PT_0 shall be assigned the value of the CT from the previous cycle, i.e., ($PT1_0$, $PT2_0$, . . ., $PT128_0$) = ($CT1_{9998}$, $CT2_{9998}$, . . ., $CT128_{9998}$). (Note that the new PT shall be denoted as PT_0 because this value is used for the first pass through the inner loop when j=0.)

NOTE: The output for this test shall consist of 400 sets of (i, KEY, IV, PT, CT). At the beginning of each of the 400 loops for i, the chaining value CV_0 shall be recorded in the IV position for the set (see the example file, "`cbc_e_m.txt`"). Essentially, the value of IV_i equals the final value of CV_0 from loop i-1.

4.4 CBC Decrypt MCT

```
Initialize  KEY0, IV0, CT0

FOR i = 0 TO 399
 {
        If (i==0) CV0 = IV0
        Record i, KEYi, CV0, CT0
        FOR j = 0 TO 9,999
         {
                IBj = CTj
                Perform algorithm in decrypt state, resulting in OBj
                PTj = OBj ⊕ CVj
                CVj+1 = CTj
                CTj+1 = PTj
         }
        Record PTj

        KEYi+1 = KEYi ⊕ last n bits of PT, where n=128, 192, or 256 (depending on key size)
        CV0 = CT9999
        CT0 = PT9999
 }
```

Figure 6: *Monte Carlo Test - CBC Decryption*

As summarized in Figure 6, the Monte Carlo Test for the CBC Decrypt state shall be performed as follows.

1. The submitter shall initialize KEY, the initialization vector (IV) and ciphertext (CT) variables. The CT and IV shall consist of 128 bits each while the KEY length shall be either 128, 192, or 256 bits.

2. The submitter shall perform the following for i=0 through 399:
   a. If i=0 (if this is the first time through this loop), set the chaining value CV_0 equal to the IV.
   b. Record the current values of the outer loop number i, KEY_i, CV_0, and CT_0.
   c. Perform the following for j=0 through 9999:
      i. Set the input block IB_j equal to the value of CT_j, i.e., ($IB1_j$, $IB2_j$, . . ., $IB128_j$) = ($CT1_j$, $CT2_j$, . . ., $CT128_j$).
      ii. Process the IB_j through the candidate algorithm in the decrypt state, resulting in an output block OB_j.
      iii. Form the plaintext PT_j by exclusive-ORing OB_j with the current CV_j, i.e., ($PT1_j$, $PT2_j$, . . ., $PT128_j$) = ($OB1_j \oplus CV1_j$, $OB2_j \oplus CV2_j$, . . ., $OB128_j \oplus CV128_j$).
      iv. Prepare for the j+1 loop by doing the following:

- Assign CV_{j+1} with the value of the current CT_j, i.e., ($CV1_{j+1}$, $CV2_{j+1}$, . . ., $CV128_{j+1}$) = ($CT1_j$, $CT2_j$, . . ., $CT128_j$);
- Assign CT_{j+1} with the value of the current PT_j, i.e., ($CT1_{j+1}$, $CT2_{j+1}$, . . ., $CT128_{j+1}$) = ($PT1_j$, $PT2_j$, . . .,$PT128_j$).

d. Record PT_j.

e. Assign a new value to the KEY in preparation for the next outer loop. The new KEY shall be calculated by exclusive-ORing the current KEY with the current PT. For example if the size of KEY is 128 bits, this equates to ($KEY1_{i+1}$, $KEY2_{i+1}$, . . ., $KEY128_{i+1}$) = ($KEY1_i \oplus PT1_{9999}$, $KEY2_i \oplus PT2_{9999}$, . . ., $KEY128_i \oplus PT128_{9999}$).

For cases when the size of KEY is 192 or 256 bits, PT shall be expanded in length to 192 or 256 bits (as appropriate) before the new KEY can be formed. This expansion shall be accomplished by concatenating the 64 or 128 rightmost bits of the previous PT (PT_{9998}) with the 128 bits of the current PT (PT_{9999}). This value shall then be exclusive-ORed with the current KEY to form the new KEY; e.g., if the size of KEY is 192 bits, ($KEY1_{i+1}$, $KEY2_{i+1}$, . . ., $KEY192_{i+1}$) = ($KEY1_i \oplus PT65_{9998}$, $KEY2_i \oplus PT66_{9998}$, . . ., $KEY64_i \oplus PT128_{9998}$, $KEY65_i \oplus PT1_{9999}$, $KEY66_i \oplus PT2_{9999}$, . . ., $KEY192_i \oplus PT128_{9999}$).

f. Assign a new value to CV_0 in preparation for the next outer loop. CV_0 shall be assigned the value of the current CT, i.e., ($CV1_0$, $CV2_0$, . . ., $CV128_0$) = ($CT1_{9999}$, $CT2_{9999}$, . . ., $CT128_{9999}$). (Note that the new CV shall be denoted as CV_0 to be used for the first pass through the inner loop when j=0.)

g. Assign a new value to CT in preparation for the next outer loop. CT_0 shall be assigned the value of the current PT, i.e., ($CT1_0$, $CT2_0$, . . ., $CT128_0$) = ($PT1_{9999}$, $PT2_{9999}$, . . ., $PT128_{9999}$). (Note that the new CT shall be denoted as CT_0 to be used for the first pass through the inner loop when j=0.)

NOTE: The output for this test shall consist of 400 sets of (i, KEY, IV, CT, PT). At the beginning of each of the 400 loops for i, the chaining value CV_0 shall be recorded in the IV position for the set (see the example file, "`cbc_d_m.txt`"). Essentially, the value of IV_i equals the final value of CV_0 from the loop i-1.

5. Summary of Required Test Value Sets

Note that there shall be a separate set of values within a particular file for each of the key sizes listed below. "Filename" indicates the name of the file in which the submitted values shall be contained. See the example files for the required formatting and syntax of submitted test values.

Known Answer Test values:

Filename	Mode	Test	Key Sizes *(bits)*
`ecb_vk.txt`	ECB	Variable Key KAT	128, 192, 256
`ecb_vt.txt`	ECB	Variable Text KAT	128, 192, 256
`ecb_tbl.txt` *(if applicable)*	ECB	Tables KAT	128, 192, 256
? (possibly multiple files) *(if applicable)*	ECB	Intermediate Values KAT	128, 192, 256

Monte Carlo Test values:

Filename	Mode	Test	Key Sizes *(bits)*
`ecb_e_m.txt`	ECB	Encrypt MCT	128, 192, 256
`ecb_d_m.txt`	ECB	Decrypt MCT	128, 192, 256
`cbc_e_m.txt`	CBC	Encrypt MCT	128, 192, 256
`cbc_d_m.txt`	CBC	Decrypt MCT	128, 192, 256

6. References

[MOVS] Draft NIST Special Publication 800-17, *Modes of Operation Validation System (MOVS): Requirements and Procedures*, May 1996.