Specifications of Cryptographic Algorithm Implementation Testing — Symmetric-Key Cryptography —

April 1, 2009

INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN
Contents

1 Introduction 1
  1.1 Organization ................................................................. 1
  1.2 Outline of the JCATT .................................................. 1

2 Scope 2
  2.1 Block Ciphers ................................................................. 2
  2.2 Stream Ciphers .............................................................. 2

3 The Tests Description of Block Ciphers 3
  3.1 Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text) ............. 4
  3.2 Variable Key Known Answer Test (KAT-Key) .................................. 5
  3.3 Multi-block Message Test (MMT) ............................................. 7
  3.4 Monte Carlo Test (MCT) ..................................................... 7
    3.4.1 MCT for 3-key Triple DES ........................................... 7
      3.4.1.1 Monte Carlo Test – ECB ......................................... 7
      3.4.1.2 Monte Carlo Test – CBC ......................................... 8
      3.4.1.3 Monte Carlo Test – CFB ......................................... 9
      3.4.1.4 Monte Carlo Test – CFB-1 ....................................... 9
      3.4.1.5 Monte Carlo Test – CFB-8 ....................................... 10
      3.4.1.6 Monte Carlo Test – OFB ......................................... 11
      3.4.1.7 Monte Carlo Test – CTR ......................................... 12
    3.4.2 MCT for 64-bit Block Cipher(not 3-key Triple DES) ......................... 12
      3.4.2.1 Monte Carlo Test – ECB ......................................... 12
      3.4.2.2 Monte Carlo Test – CBC ......................................... 13
      3.4.2.3 Monte Carlo Test – CFB ......................................... 14
      3.4.2.4 Monte Carlo Test – OFB ......................................... 15
      3.4.2.5 Monte Carlo Test – CTR ......................................... 16
    3.4.3 MCT for 128-bit Block Cipher .......................................... 16
      3.4.3.1 Monte Carlo Test – ECB ......................................... 16
      3.4.3.2 Monte Carlo Test – CBC ......................................... 17
      3.4.3.3 Monte Carlo Test – CFB ......................................... 18
      3.4.3.4 Monte Carlo Test – CFB-1(Only for AES) ............................. 19
      3.4.3.5 Monte Carlo Test – CFB-8(Only for AES) ............................ 21
      3.4.3.6 Monte Carlo Test – OFB ......................................... 22
      3.4.3.7 Monte Carlo Test – CTR ......................................... 23
  3.5 Sbox Known Answer Test (KAT-Sbox) ...................................... 24
  3.6 Other KATs for 3-key Triple DES ......................................... 24

4 The Tests Description of Stream Ciphers 25
  4.1 MUGI .......................................................... 25
    4.1.1 Variable Key Known Answer Test (KAT-Key) ............................. 25
    4.1.2 Monte Carlo Test (MCT) ............................................. 25
  4.2 MULTI-S01 ................................................... 25
    4.2.1 Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text) .......... 25
    4.2.2 Variable Key Known Answer Test (KAT-Key) ............................. 26
    4.2.3 Monte Carlo Test (MCT) ............................................. 26
  4.3 128-bit RC4 ................................................... 26
    4.3.1 Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text) .......... 26
    4.3.2 Variable Key Known Answer Test (KAT-Key) ............................. 27
    4.3.3 Monte Carlo Test (MCT) ............................................. 27

5 Conditions for Issuing Cryptographic Algorithm Validation Certificate 28
  5.1 Details of Conditions .................................................... 28
1 Introduction

This document describes the specifications of cryptographic algorithm implementation testing of the symmetric-key cryptographic algorithms.

1.1 Organization

Section 2 specifies the cryptographic algorithms that are in the scope of this document. Section 3 provides an overview of the tests of each block cipher algorithm that make up the JCATT. Section 4 provides an overview of the tests of each stream cipher algorithm that make up the JCATT. Section 5 provides the conditions for issuing cryptographic algorithm validation certificate.

The following acronyms are used throughout this document.

- JCATT: Japan Cryptographic Algorithm implementation Testing Tool
- IUT: Implementation Under Test

1.2 Outline of the JCATT

The Japan Cryptographic Algorithm implementation Testing Tool is designed:

- to test conformance to the cryptographic algorithm specifications,
- To test each function of the cryptographic algorithm — for example, encryption, decryption, etc for symmetric-key cryptography. —
- to allow the testing of an IUT at locations remote to the JCATT. The JCATT and the IUT communicate data via REQUEST and RESPONSE files.

Once configuration information has been provided, appropriate REQUEST files will be generated. REQUEST files are the means by which test data is communicated to the IUT. The IUT is used to process the data in the REQUEST file, and the resulting data is placed in a RESPONSE file. The data in the RESPONSE file is then verified.

The specification of the file format is available in ref. [4] and the sample files are available in ref. [5].

Figure 1.1: The Workflow of the Cryptographic Algorithm Implementation Testing

![Diagram of the Workflow of the Cryptographic Algorithm Implementation Testing]
2 Scope

This document specifies the tests required to validate IUTs for conformance to the specifications of the following cryptographic algorithms.

2.1 Block Ciphers

64-bit block cipher
- CIPHERUNICORN-E
- Hierocrypt-L1
- MISTY1
- 3-key Triple DES

128-bit block cipher
- AES
- Camellia
- CIPHERUNICORN-A
- Hierocrypt-3
- SC2000

2.2 Stream Ciphers

- MUGI
- MULTI-S01
- 128-bit RC4
3 The Tests Description of Block Ciphers

This section provides the tests description of the following 64-bit block cipher algorithms:

- CIPHERUNICORN-E
- Hierocrypt-L1
- MISTY1
- 3-key Triple DES

and the following 128-bit block cipher algorithms:

- AES
- Camellia
- CIPHERUNICORN-A
- Hierocrypt-3
- SC2000

for the following Modes of Operation:

- ECB
- CBC
- CFB
- OFB
- CTR

In OFB and CFB mode, the length of the data segment is equal to the cipher block size.

For AES and 3-key Triple DES, the following Modes of Operation may be tested:

- CFB-1 (CFB where the length of the data segment is 1 bit, s=1)
- CFB-8 (CFB where the length of the data segment is 8 bits, s=8)

CTR mode shall be implemented using the standard incrementing function \( x \leftarrow x + 1 \mod 2^m \), as referred in Appendix B.1 of NIST SP 800-38A[3], for generation of counter blocks.

For each 128-bit block ciphers (AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, and SC2000), selections are available for the key sizes (i.e., 128-bit, 192-bit, and 256-bit.)

The tests description of each block cipher algorithm is shown below.

### AES

- Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text)
- Variable Key Known Answer Test (KAT-Key)
- Multi-block Message Test (MMT)
- Monte Carlo Test (MCT)
- GFSbox Known Answer Test (KAT-GFSbox)
- KeySbox Known Answer Test (KAT-KeySbox)

### 3-key Triple DES

- Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text)
- Variable Key Known Answer Test (KAT-Key)
- Multi-block Message Test (MMT)
- Monte Carlo Test (MCT)
- Inverse Permutation Known Answer Test (KAT-IP)
- Permutation Operation Known Answer Test (KAT-PO)
- Substitution Table Known Answer Test (KAT-ST)
Other block ciphers

- Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text)
- Variable Key Known Answer Test (KAT-Key)
- Multi-block Message Test (MMT)
- Monte Carlo Test (MCT)
- Sbox Known Answer Test (KAT-Sbox)

The test items of AES are based on AESAVS[1], and the test items of 3-key Triple DES are based on TMOVS[2], although parameters of each test item (for example, the number of iteration of loops in MCT) are defined in section 5.

The test vectors for each KAT for AES and 3-key Triple DES are ones described in AESAVS and TMOVS respectively.

The test items for cryptographic algorithms other than AES and 3-key Triple DES are based on AESAVS[1].

3.1 Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text)

To perform the KAT-text, JCATT supplies the IUT with initial values for the key, the plaintext(s) (ciphertext(s)) and, if applicable, the initialization vector(s). The key is initialized to zero (with odd parity set for 3-Key Triple DES.)

The plaintext(or IV, counter) for KAT-Text described in AESAVS:

\begin{verbatim}
0x80000000 00000000 00000000 00000000
0xc0000000 00000000 00000000 00000000
0xe0000000 00000000 00000000 00000000
0xf0000000 00000000 00000000 00000000
0xf8000000 00000000 00000000 00000000
0xfc000000 00000000 00000000 00000000
0xfe000000 00000000 00000000 00000000
0xff000000 00000000 00000000 00000000
0xff800000 00000000 00000000 00000000
0xffc00000 00000000 00000000 00000000
0xffe00000 00000000 00000000 00000000
0xfff00000 00000000 00000000 00000000
\ldots
0xffffffff ffffffff ffffffff ffffffff
0xffffffff ffffffff ffffffff ffffffff
\end{verbatim}

The plaintext(or IV, counter) for KAT-Text described in TMOVS:

\begin{verbatim}
0x80000000 00000000
0xc0000000 00000000
0xe0000000 00000000
0xf0000000 00000000
0xf8000000 00000000
0xfc000000 00000000
0xfe000000 00000000
0xff000000 00000000
0xff800000 00000000
0xffc00000 00000000
0xffe00000 00000000
0xfff00000 00000000
\ldots
0x00000000 00000002
\end{verbatim}
the test vectors for each mode of operation are shown in Table 3.1, 3.2, and 3.3. The test vectors for CTR are the same as those for OFB.
In AESAVS, the input ciphertexts for KAT-Text decryption are the expected ciphertexts in the KAT-Text encryption.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Plaintext</th>
<th>IV or Counter</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>The test vector shown above</td>
<td>—</td>
<td>all-0</td>
</tr>
<tr>
<td>CBC</td>
<td>The test vector shown above</td>
<td>all-0</td>
<td>all-0</td>
</tr>
<tr>
<td>CFB128</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>all-0</td>
</tr>
<tr>
<td>CFB1</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>all-0</td>
</tr>
<tr>
<td>CFB8</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>all-0</td>
</tr>
<tr>
<td>OFB</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>all-0</td>
</tr>
<tr>
<td>CTR</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>all-0</td>
</tr>
</tbody>
</table>

Table 3.2: Test vectors of TMOVS: KAT-Text Encryption

<table>
<thead>
<tr>
<th>Mode</th>
<th>Plaintext</th>
<th>IV or Counter</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>The test vector shown above</td>
<td>—</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CBC</td>
<td>The test vector shown above</td>
<td>all-0</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CFB64</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CFB1</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CFB8</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>OFB</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CTR</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
</tbody>
</table>

Table 3.3: Test Vectors of TMOVS: KAT-Text Decryption

<table>
<thead>
<tr>
<th>Mode</th>
<th>Ciphertext</th>
<th>IV or Counter</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>Ciphertext from KAT-Text Encryption</td>
<td>—</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CBC</td>
<td>Ciphertext from KAT-Text Encryption</td>
<td>all-0</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CFB64</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CFB1</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CFB8</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>OFB</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
<tr>
<td>CTR</td>
<td>all-0</td>
<td>The test vector shown above</td>
<td>0101 · · · 01(Odd Parity)</td>
</tr>
</tbody>
</table>

3.2 Variable Key Known Answer Test (KAT-Key)

To implement the Variable Key Known Answer Test, JCATT supplies the IUT with initial values for the key, the plaintext(s), and, if applicable, the initialization vector(s).

The key is initialized to a vector which contains a “1” in the i-th significant position and “0”’s in all remaining significant positions. For 3-Key Triple DES, odd parity is set in addition to the vector described above.

The Input Keys for KAT-Key Described in AESAVS:

0x80000000 00000000 00000000 00000000
0xc0000000 00000000 00000000 00000000
0xe0000000 00000000 00000000 00000000
0xf0000000 00000000 00000000 00000000
0xf8000000 00000000 00000000 00000000

5/31
The Keys of TMOVS KAT-Key:

0x8001010101010101 8001010101010101 8001010101010101
0x4001010101010101 4001010101010101 4001010101010101
0x2001010101010101 2001010101010101 2001010101010101
0x1001010101010101 1001010101010101 1001010101010101
0x0801010101010101 0801010101010101 0801010101010101
0x0401010101010101 0401010101010101 0401010101010101
0x0201010101010101 0201010101010101 0201010101010101
0x0180010101010101 0180010101010101 0180010101010101
0x0140010101010101 0140010101010101 0140010101010101
0x0120010101010101 0120010101010101 0120010101010101
0x0110010101010101 0110010101010101 0110010101010101
0x0108010101010101 0108010101010101 0108010101010101
0x0101010101010104 0101010101010104 0101010101010104
0x0101010101010102 0101010101010102 0101010101010102

The test vectors for each mode of operation are shown in the table 3.4, 3.5 and 3.6. The test vectors for CTR mode are the same as those for OFB mode.

For KAT-Key for decryption, the input ciphertexts are the expected ciphertexts in the KAT-Key for encryption.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Plaintext</th>
<th>IV or counter</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>all-0</td>
<td>—</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CBC</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB128</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB1</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB8</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>OFB</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CTR</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode</th>
<th>Plaintext</th>
<th>IV or counter</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>all-0</td>
<td>—</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CBC</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB64</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB1</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB8</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>OFB</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CTR</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
</tbody>
</table>
### 3.3 Multi-block Message Test (MMT)

The Multi-block Message Test (MMT) is designed to test the ability of the implementation to process multi-block messages, which may require chaining of information from one block to the next. The test supplies the IUT with messages that are integral numbers of blocks in length. For ECB, CBC, OFB, and CFB128 the block length is 128 bits, for CFB1 the block length is 1 bit, and for CFB8 the block length is 8 bits. The default number of MMT blocks is defined in section 5.

### 3.4 Monte Carlo Test (MCT)

Each Monte Carlo Test ciphers a certain number of pseudorandom texts. These texts are generated using the algorithm specified in the section below pertaining to the mode of operation being tested. For modes that use an IV, the IV is used at the beginning of each pseudorandom text. The default number of the iterations of inner loop and the iterations of outer loop are defined in section 5.

#### 3.4.1 MCT for 3-key Triple DES

MCT for 3-key Triple DES is similar to the algorithm described in TMOVS. MCT for CTR mode (incremental counter), which is not described in TMOVS, is based on MCT for ECB mode. The default number of loop iterations is defined in section 5.

##### 3.4.1.1 Monte Carlo Test – ECB

**ECB Encryption**

```plaintext
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
PT = PT_0 // Initial plaintext

for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], PT) // ECB Encryption
    PT = CT[j]
  }

Output CT[innerloop-1] // Last ciphertext computed in the inner loop

Key1[i+1] = Key1[i] xor CT[innerloop-1]
Key2[i+1] = Key2[i] xor CT[innerloop-2]
Key3[i+1] = Key3[i] xor CT[innerloop-3]
```

**ECB Decryption**

---

### Table 3.6: TMOVS Test Vectors: KAT-Key Decryption

<table>
<thead>
<tr>
<th>Mode</th>
<th>Ciphertext</th>
<th>IV or counter</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>Ciphertext from KAT-Key Encryption</td>
<td>—</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CBC</td>
<td>Ciphertext from KAT-Key Encryption</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB64</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB1</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CFB8</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>OFB</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
<tr>
<td>CTR</td>
<td>all-0</td>
<td>all-0</td>
<td>The test vector shown above</td>
</tr>
</tbody>
</table>
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key1[i], Key2[i], Key3[i], CT) // ECB Decryption
        CT = PT[j]
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    Key1[i+1] = Key1[i] xor PT[innerloop-1]
    Key2[i+1] = Key2[i] xor PT[innerloop-2]
    Key3[i+1] = Key3[i] xor PT[innerloop-3]
}

3.4.1.2 Monte Carlo Test – CBC

3.4.1.2.1 CBC Encryption

Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // CBC Encryption
        if (i==0)
        {
            PT = IV
        } else
        {
            PT = CT[j-1]
        }
        IV = CT[j]
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    Key1[i+1] = Key1[i] xor CT[innerloop-1]
    Key2[i+1] = Key2[i] xor CT[innerloop-2]
    Key3[i+1] = Key3[i] xor CT[innerloop-3]
}

3.4.1.2.2 CBC Decryption

Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key1[i], Key2[i], Key3[i], IV, CT) // CBC decryption
        if (i==0)
        {
            IV = CT
            CT = PT[j]
        }
        Output PT[innerloop-1] // Last plaintext computed in the inner loop
3.4.1.3 Monte Carlo Test – CFB

3.4.1.3.1 CFB Encryption

```
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV(64-bit)
PT = PT_0 // Initial plaintext(64-bit)
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // CFB encryption
    PT = IV
    IV = CT[j]
  }
  Output CT[innerloop-1] // Last ciphertext computed in the inner loop
  Key1[i+1] = Key1[i] xor CT[innerloop-1]
  Key2[i+1] = Key2[i] xor CT[innerloop-2]
  Key3[i+1] = Key3[i] xor CT[innerloop-3]
}
```

3.4.1.3.2 CFB Decryption

```
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV(64-bit)
CT = CT_0 // Initial ciphertext(64-bit)
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key1[i], Key2[i], Key3[i], IV, CT) // CFB decryption
    IV = CT
    CT = 0[j] // 0[j] = The value right before xor-ing ciphertext in Decryption()
  }
  Output PT[innerloop-1] // Last plaintext computed in the inner loop
  Key1[i+1] = Key1[i] xor PT[innerloop-1]
  Key2[i+1] = Key2[i] xor PT[innerloop-2]
  Key3[i+1] = Key3[i] xor PT[innerloop-3]
}
```

3.4.1.4 Monte Carlo Test – CFB-1

3.4.1.4.1 CFB-1 Encryption

```
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV(64-bit)
PT = PT_0 // Initial plaintext(1-bit)
for (i=0; i<outerloop; i++)
{
```

9/31
for (j=0; j<innerloop; j++)
{
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // CFB-1 Encryption
    PT = Most significant bit of IV
    IV = (rightmost 63 bits of IV) || CT[j]
}
Output CT[innerloop-1] // Last ciphertext computed in the inner loop
// Concatenate ciphertexts to make it 192-bit
C = CT[innerloop-192]||...||CT[innerloop-2]||CT[innerloop-1]
Key1[i+1] = Key1[i] xor (rightmost 64 bits of C)
Key2[i+1] = Key2[i] xor (middle 64 bits of C)
Key3[i+1] = Key3[i] xor (leftmost 64 bits of C)
}

3.4.1.4.2 CFB-1 Decryption

Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV(64-bit)
CT = CT_0 // Initial ciphertext(1-bit)
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key1[i], Key2[i], Key3[i], IV, CT) // CFB-1 decryption
        IV = (rightmost 63 bits of IV) || CT
        // O[j] = The value right before xor-ing ciphertext in Decryption()
        CT = (Most Significant Bit of O[j])
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    // Concatenate the plaintext to make it 192-bit
    P = PT[innerloop-192]||...||PT[innerloop-2]||PT[innerloop-1]
    Key1[i+1] = Key1[i] xor (rightmost 64 bits of P)
    Key2[i+1] = Key2[i] xor (middle 64 bits of P)
    Key3[i+1] = Key3[i] xor (leftmost 64 bits of P)
}

3.4.1.5 Monte Carlo Test – CFB-8

3.4.1.5.1 CFB-8 Encryption

Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV(64-bit)
PT = PT_0 // Initial plaintext(8-bit)
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // CFB-8 encryption
        PT = Leftmost 8 bits of IV
        IV = (Rightmost 56 bits of IV) || CT[j]
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    // Concatenate the ciphertexts to make it 192-bit
    C = CT[innerloop-24]||...||CT[innerloop-2]||CT[innerloop-1]
    Key1[i+1] = Key1[i] xor (Rightmost 64 bits of C)
    Key2[i+1] = Key2[i] xor (Middle 64 bits of C)
    Key3[i+1] = Key3[i] xor (Leftmost 64 bits of C)
}
### 3.4.1.5.2 CFB-8 Decryption

```
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV(64-bit)
CT = CT_0 // Initial ciphertext(8-bit)
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key1[i], Key2[i], Key3[i], IV, CT) // CFB-8 decryption
        IV = (Rightmost 56 bits of IV) || CT
        // O[j] = The value right before xor-ing ciphertext in Decryption()
        CT = (Leftmost 8 bits of O[j])
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    // Concatenate the plaintexts to make it 192-bit
    P = PT[innerloop-24]||...||PT[innerloop-2]||PT[innerloop-1]
    Key1[i+1] = Key1[i] xor (Rightmost 64 bits of P)
    Key2[i+1] = Key2[i] xor (Middle 64 bits of P)
    Key3[i+1] = Key3[i] xor (Leftmost 64 bits of P)
}
```

### 3.4.1.6 Monte Carlo Test – OFB

#### 3.4.1.6.1 OFB Encryption

```
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    INIT_PT = PT
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // OFB encryption
        PT = IV
        IV = O[j] // O[j] = The value right before xor-ing plaintext in Encryption()
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    Key1[i+1] = Key1[i] xor CT[innerloop-1]
    Key2[i+1] = Key2[i] xor CT[innerloop-2]
    Key3[i+1] = Key3[i] xor CT[innerloop-3]
    PT = PT xor INIT_PT
}
```

#### 3.4.1.6.2 OFB Decryption

```
Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
IV = IV_0 // Initial IV
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
    INIT_CT = CT
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key1[i], Key2[i], Key3[i], IV, CT) // OFB decryption
    }
```
CT = IV // 0[j] = The value right before xor-ing ciphertext in Decryption()

Output PT[innerloop-1] // Last plaintext computed in the inner loop
Key1[i+1] = Key1[i] xor PT[innerloop-1]
Key2[i+1] = Key2[i] xor PT[innerloop-2]
Key3[i+1] = Key3[i] xor PT[innerloop-3]
CT = CT xor INIT_CT

3.4.1.7 Monte Carlo Test – CTR

3.4.1.7.1 CTR Encryption

Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
CTR = CTR_0 // Initial Counter
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], CTR, PT) // CTR Encryption
    CTR = (CTR + 1) mod 2^64
    PT = CT[j]
  }
  Output CT[innerloop-1] // Last ciphertext computed in the inner loop
  Key1[i+1] = Key1[i] xor CT[innerloop-1]
  Key2[i+1] = Key2[i] xor CT[innerloop-2]
  Key3[i+1] = Key3[i] xor CT[innerloop-3]
}

3.4.1.7.2 CTR Decryption

Key1[0] = Key1 // Initial Key(64-bit)
Key2[0] = Key2 // Initial Key(64-bit)
Key3[0] = Key3 // Initial Key(64-bit)
CTR = CTR_0 // Initial Counter
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key1[i], Key2[i], Key3[i], CTR, CT) // CTR decryption
    CTR = (CTR + 1) mod 2^64
    CT = PT[j]
  }
  Output PT[innerloop-1] // Last plaintext computed in the inner loop
  Key1[i+1] = Key1[i] xor PT[innerloop-1]
  Key2[i+1] = Key2[i] xor PT[innerloop-2]
  Key3[i+1] = Key3[i] xor PT[innerloop-3]
}

3.4.2 MCT for 64-bit Block Cipher(not 3-key Triple DES)

3.4.2.1 Monte Carlo Test – ECB

3.4.2.1.1 ECB Encryption
Key[0] = Key // Initial Key
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], PT) // ECB encryption
        PT = CT[j]
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}

3.4.2.1.2 ECB Decryption

Key[0] = Key // Initial Key
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key[i], CT) // ECB decryption
        CT = PT[j]
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
}

3.4.2.2 Monte Carlo Test – CBC

3.4.2.2.1 CBC Encryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], IV, PT) // CBC encryption
        if ( j==0 )
        {
            PT = IV
        }
        else
        {
            PT = CT[j-1]
        }
        IV = CT[j]
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}

3.4.2.2.2 CBC Decryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{ 
  for (j=0; j<innerloop; j++)
  {  
    PT[j] = Decryption(Key[i], IV, CT) // CBC decryption
      If( j==0 ) 
      { 
        tmp = CT
        CT = IV
        IV = tmp
      } else 
      { 
        IV = CT
        CT = PT[j-1]
      } 
  } 
  Output PT[innerloop-1] // Last plaintext computed in the inner loop
  Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  IV = PT[innerloop-1]
}

3.4.2.3 Monte Carlo Test – CFB

3.4.2.3.1 CFB Encryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{ 
  for (j=0; j<innerloop; j++)
  {  
    CT[j] = Encryption(Key[i], IV, PT) // CFB encryption
      If( j==0 ) 
      { 
        PT = IV
      } else 
      { 
        PT = CT[j-1]
      } 
    IV = CT[j]
  } 
  Output CT[innerloop-1] // Last ciphertext computed in the inner loop
  Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}

3.4.2.3.2 CFB Decryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{ 
  for (j=0; j<innerloop; j++)
  {  
    PT[j] = Decryption(Key[i], IV, CT) // CFB decryption
      If( j==0 ) 
      { 
        tmp = IV
        IV = CT
        CT = tmp
      } else 
      { 
      } 
}
3.4.2.4 Monte Carlo Test – OFB

3.4.2.4.1 OFB Encryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], IV, PT) // OFB encryption
        If( j==0 )
            { PT = IV }
        else
            { PT = CT[j-1] }
        IV = O[j] // O[j] = The value right before xor-ing plaintext in Encryption()
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
    IV = CT[innerloop-1]
}

3.4.2.4.2 OFB Decryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key[i], IV, CT) // OFB decryption
        If( j==0 )
            { CT = IV }
        else
            { CT = PT[j-1] }
        IV = O[j] // O[j] = The value right before xor-ing ciphertext in Decryption()
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
    IV = PT[innerloop-1]
### 3.4.2.5 Monte Carlo Test – CTR

#### 3.4.2.5.1 CTR Encryption

```plaintext
Key[0] = Key // Initial Key
CTR = CTR_0 // Initial counter
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], CTR, PT) // CTR encryption
        CRT = (CTR + 1) mod 2^64
        PT = CT[j]
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}
```

#### 3.4.2.5.2 CTR Decryption

```plaintext
Key[0] = Key // Initial Key
CTR = CTR_0 // Initial counter
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key[i], CTR, CT) // CTR decryption
        CRT = (CTR + 1) mod 2^64
        CT = PT[j]
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
}
```

### 3.4.3 MCT for 128-bit Block Cipher

#### 3.4.3.1 Monte Carlo Test – ECB

#### 3.4.3.1.1 ECB Encryption

```plaintext
Key[0] = Key // Initial Key
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], PT) // ECB encryption
        PT = CT[j]
    }
    Output CT[innerloop-1]
    If ( KeyLength == 128-bit )
    {
        Key[i+1] = Key[i] xor CT[innerloop-1]
    }
    If ( KeyLength == 192-bit )
    {
        Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
    }
    If ( KeyLength == 256-bit )
    {
```
3.4.3.1.2 ECB Decryption

Key[0] = Key // Initial Key
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
   for (j=0; j<innerloop; j++)
   {
      PT[j] = Decryption(Key[i], CT) // ECB decryption
      CT = PT[j]
   }
   Output PT[innerloop-1]
   If ( KeyLength == 128-bit )
   {
      Key[i+1] = Key[i] xor PT[innerloop-1]
   }
   If ( KeyLength == 192-bit )
   {
      Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
   }
   If ( KeyLength == 256-bit )
   {
      Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
   }
}

3.4.3.2 Monte Carlo Test – CBC

3.4.3.2.1 CBC Encryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
   for (j=0; j<innerloop; j++)
   {
      CT[j] = Encryption(Key[i], IV, PT) // CBC encryption
      If( j==0 )
      {
         PT = IV
      }else
      {
         PT = CT[j-1]
      }
      IV = CT[j]
   }
   Output CT[innerloop-1] // Last ciphertext computed in the inner loop
   If ( KeyLength ==128-bit )
   {
      Key[i+1] = Key[i] xor CT[innerloop-1]
   }
   If ( KeyLength == 192-bit )
   {
      Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
   }
   If ( KeyLength == 256-bit )
   {
      Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
   }
}
3.4.3.2 CBC Decryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key[i], IV, CT) // CBC decryption
        If( j==0 )
        {
            tmp = CT
            CT = IV
            IV = tmp
        } else
        {
            IV = CT
            CT = PT[j-1]
        }
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    If ( KeyLength ==128-bit )
    {
        Key[i+1] = Key[i] xor PT[innerloop-1]
    }
    If ( KeyLength == 192-bit )
    {
        Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
    }
    If ( KeyLength == 256-bit )
    {
        Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
    }
    IV = PT[innerloop-1] // Last plaintext computed in the inner loop
}

3.4.3.3 Monte Carlo Test – CFB

3.4.3.3.1 CFB Encryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV(128-bit)
PT = PT_0 // Initial plaintext(128-bit)
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], IV, PT) // CFB encryption
        If( j==0 )
        {
            PT = IV
        } else
        {
            PT = CT[j-1]
        }
        IV = CT[j]
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    If ( KeyLength ==128-bit )

3.4.3.3.2 CFB Decryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV(128-bit)
CT = CT_0 // Initial ciphertext(128-bit)
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key[i], IV, CT) // CFB decryption
        If( j==0 )
        {
            tmp = IV
            IV = CT
            CT = tmp
        } else
        {
            IV = CT
            CT = PT[j-1]
        }
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    If ( KeyLength == 128-bit )
    {
        Key[i+1] = Key[i] xor PT[innerloop-1]
    }
    If ( KeyLength == 192-bit )
    {
        Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
    }
    If ( KeyLength == 256-bit )
    {
        Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
    }
    IV = PT[innerloop-1]
}

3.4.3.4 Monte Carlo Test – CFB-1(Only for AES)

3.4.3.4.1 CFB-1 Encryption

Key[0] = Key // Initial Key
IV[0] = IV_0 // Initial IV(128-bit)
PT = PT_0 // Initial plaintext(1-bit)
for (i=0; i<outerloop; i++)
{
    IV = IV[i]
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], IV, PT) // CFB-1 encryption
    }
}
IV = (Rightmost 127 bits of IV) || CT[j]
if( j<128 )
   {  
   PT = (Most Significant Bit of IV[i])
   }
else
   {  
   PT = CT[j-128]
   }
Output CT[innerloop-1] // Last ciphertext computed in the inner loop
If ( KeyLength == 128-bit )
   {
   Key[i+1] = Key[i] xor CT[innerloop-128] || CT[innerloop-127] || ...
   || CT[innerloop-1]
   }
If ( KeyLength == 192-bit )
   {
   Key[i+1] = Key[i] xor CT[innerloop-192] || CT[innerloop-191] || ...
   || CT[innerloop-1]
   }
If ( KeyLength == 256-bit )
   {
   Key[i+1] = Key[i] xor CT[innerloop-256] || CT[innerloop-255] || ...
   || CT[innerloop-1]
   }
IV[i+1] = CT[innerloop-128] || CT[innerloop-127] || ... || CT[innerloop-1]

3.4.3.4.2 CFB-1 Decryption

Key[0] = Key // Initial Key
IV[0] = IV_0 // Initial IV(128-bit)
CT = CT_0 // Initial ciphertext(1-bit)
for (i=0; i<outerloop; i++)
   {
   IV = IV[i]
   for (j=0; j<innerloop; j++)
      {
      PT[j] = Decryption(Key[i], IV, CT) // CFB-1 decryption
      IV = (Rightmost 127 bits of IV) || CT
      if( j<128 )
         {
         CT = j-th significant bit of IV[i]
         }
      else
         {
         CT = PT[j-128]
         }
      }
Output PT[innerloop-1] // Last plaintext computed in the inner loop
If ( KeyLength == 128-bit )
   {
   Key[i+1] = Key[i] xor PT[innerloop-128] || PT[innerloop-127] || ...
   || PT[innerloop-1]
   }
If ( KeyLength == 192-bit )
   {
   Key[i+1] = Key[i] xor PT[innerloop-192] || PT[innerloop-191] || ...
   || PT[innerloop-1]
   }
If ( KeyLength == 256-bit )
   {
   Key[i+1] = Key[i] xor PT[innerloop-256] || PT[innerloop-255] || ...
   || PT[innerloop-1]
   }
IV[i+1] = PT[innerloop-128] || PT[innerloop-127] || ... || PT[innerloop-1]
3.4.3.5 Monte Carlo Test – CFB-8 (Only for AES)

3.4.3.5.1 CFB-8 Encryption

Key[0] = Key // Initial Key
IV[0] = IV_0 // Initial IV (128-bit)
PT = PT_0 // Initial plaintext (8-bit)
for (i=0; i<outerloop; i++)
{
    IV = IV[i]
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], IV, PT) // CFB-8 encryption
        IV = (Rightmost 120 bits of IV) || CT[j]
        if (j<16)
            { PT = j-th significant bit of IV[i] }
        else
            { PT = CT[j-16] }
    }
    Output CT[innerloop-1] // Last ciphertext computed in the inner loop
    If (KeyLength == 128-bit)
        { Key[i+1] = Key[i] xor CT[innerloop-16] || CT[innerloop-15] || ... || CT[innerloop-1] }
    If (KeyLength == 192-bit)
        { Key[i+1] = Key[i] xor CT[innerloop-24] || CT[innerloop-23] || ... || CT[innerloop-1] }
    If (KeyLength == 256-bit)
        { Key[i+1] = Key[i] xor CT[innerloop-32] || CT[innerloop-31] || ... || CT[innerloop-1] }
    IV[i+1] = CT[innerloop-16] || CT[innerloop-15] || ... || CT[innerloop-1]
}

3.4.3.5.2 CFB-8 Decryption

Key[0] = Key // Initial Key
IV[0] = IV_0 // Initial IV (128-bit)
CT = CT_0 // Initial ciphertext (8-bit)
for (i=0; i<outerloop; i++)
{
    IV = IV[i]
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key[i], IV, CT) // CFB-8 decryption
        IV = (Rightmost 120 bits of IV) || CT
        if (j<16)
            { CT = j-th significant bit of IV[i] }
        else
            { CT = PT[j-16] }
}
Output PT[innerloop-1] // Last plaintext computed in the inner loop
If ( KeyLength == 128-bit )
{
    Key[i+1] = Key[i] xor PT[innerloop-16] || PT[innerloop-15] || ...
    || PT[innerloop-1]
}
If ( KeyLength == 192-bit )
{
    Key[i+1] = Key[i] xor PT[innerloop-24] || PT[innerloop-23] || ...
    || PT[innerloop-1]
}
If ( KeyLength == 256-bit )
{
    Key[i+1] = Key[i] xor PT[innerloop-32] || PT[innerloop-31] || ...
    || PT[innerloop-1]
}

3.4.3.6 Monte Carlo Test – OFB

3.4.3.6.1 OFB Encryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        CT[j] = Encryption(Key[i], IV, PT) // OFB encryption
        If( j==0 )
        {
            PT = IV
        }
        else
        {
            PT = CT[j-1]
        }
    }
    IV = 0[j] // 0[j] = The value right before xor-ing plaintext in Encryption()
}
Output CT[innerloop-1] // Last ciphertext computed in the inner loop
If ( KeyLength == 128-bit )
{
    Key[i+1] = Key[i] xor CT[innerloop-1]
}
If ( KeyLength == 192-bit )
{
    Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
}
If ( KeyLength == 256-bit )
{
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}
IV = CT[innerloop-1]

3.4.3.6.2 OFB Decryption

Key[0] = Key // Initial Key
IV = IV_0 // Initial IV
CT = CT_0 // Initial ciphertext
for (i=0; i<outerloop; i++)
{ 
for (j=0; j<innerloop; j++)
{
    PT[j] = Decryption(Key[i], IV, CT) // OFB decryption
    If( j==0 )
    {
        CT = IV
    }
    else
    {
        CT = PT[j-1]
    }
    IV = O[j] // O[j] = The value right before xor-ing ciphertext in Decryption()
}
Output PT[innerloop-1] // Last plaintext computed in the inner loop
If ( KeyLength == 128-bit )
{
    Key[i+1] = Key[i] xor PT[innerloop-1]
}
If ( KeyLength == 192-bit )
{
    Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
}
If ( KeyLength == 256-bit )
{
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
    IV = PT[innerloop-1]
}
}

3.4.3.7 Monte Carlo Test – CTR

3.4.3.7.1 CTR Encryption

Key[0] = Key // Initial Key
CTR = CTR_0 // Initial counter
PT = PT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
for (j=0; j<innerloop; j++)
{
    CT[j] = Encryption(Key[i], CTR, PT) // CTR encryption
    CTR = (CTR + 1) mod 2^128
    PT = CT[j]
}
Output CT[innerloop-1] // Last ciphertext computed in the inner loop
If ( KeyLength == 128-bit )
{
    Key[i+1] = Key[i] xor CT[innerloop-1]
}
If ( KeyLength == 192-bit )
{
    Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
}
If ( KeyLength == 256-bit )
{
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}
}

3.4.3.7.2 CTR Decryption

Key[0] = Key // Initial Key
CTR = CTR_0 // Initial counter
CT = CT_0 // Initial plaintext
for (i=0; i<outerloop; i++)
{
    for (j=0; j<innerloop; j++)
    {
        PT[j] = Decryption(Key[i], CTR, CT) // CTR decryption
        CTR = (CTR + 1) mod 2^128
        CT = PT[j]
    }
    Output PT[innerloop-1] // Last plaintext computed in the inner loop
    If ( KeyLength == 128-bit )
    {
        Key[i+1] = Key[i] xor PT[innerloop-1]
    }
    If ( KeyLength == 192-bit )
    {
        Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
    }
    If ( KeyLength == 256-bit )
    {
        Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
    }
}

3.5 Sbox Known Answer Test (KAT-Sbox)

To implement the KAT-Sbox Test, JCAT provides several key-data sets. The input values for the KAT-Sbox vary depending on algorithms.

The values listed in the Appendices B and C of AESAVS[1] are used for KAT-GFSbox and KAT-KeySbox for AES.

The values listed in TMOVS[2] are used for KAT-ST for 3-key Triple DES.

3.6 Other KATs for 3-key Triple DES

The values listed in TMOVS[2] are used for other KATs for 3-key Triple DES listed below:

- Inverse Permutation Known Answer Test (KAT-IP)
- Permutation Operation Known Answer Test (KAT-PO)
4 The Tests Description of Stream Ciphers

This section provides the tests description of the following stream cipher algorithms:

- MUGI
- MULTI-S01
- 128-bit RC4

While 128-bit RC4 and MULTI-S01 take plaintexts as the input, MUGI is an algorithm for pseudo-random number generation, that does not take any input plaintexts.

4.1 MUGI

As for MUGI, only pseudo-random generation is tested. There are two types of tests as follows:

- Variable Key Known Answer Test (KAT-Key)
- Monte Carlo Test (MCT)

The test vectors for KAT are based on those described in AESAVS[1]. MCT is based on the algorithm described in AESAVS[1].

4.1.1 Variable Key Known Answer Test (KAT-Key)

The same KAT-Key as KAT for AES with 128-bit key in ECB is performed. The default number of units is defined in section 5.

4.1.2 Monte Carlo Test (MCT)

MCT is performed by inspecting whether the pseudo-random number \( CT[i] \) computed by the following algorithm agrees with the expected value for every initial key and initial vector randomly given. The default numbers of the iterations of loops innerloop and outerloop are defined in section ??.

MCT for MUGI

\[
\begin{align*}
\text{Key}[0] &= \text{Key} \ // 128\text{-bit Initial Key} \\
\text{I}[0] &= \text{I} \ // \text{Initial vector} \\
n &= 2 \ // \text{The number of units} \\
\text{for} \ (j=0; \ j<\text{outerloop}; \ j++) \\
\quad \{ \\
\quad \text{for} \ (i=0; \ i<\text{innerloop}; \ i++) \\
\quad \quad \{ \\
\quad \quad \text{CT}[i] = \text{MUGI}([\text{Key}[i], \text{I}[i], n]) \\
\quad \quad \text{I}[i+1] = \text{CT}[i] \\
\quad \quad \} \\
\quad \text{Output} \ \text{CT}[i-1] \\
\quad \text{Key}[0] = \text{Key}[i-1] \oplus \text{CT}[i-1] \\
\quad \text{I}[0] = \text{CT}[i-1] \\
\\}
\end{align*}
\]

4.2 MULTI-S01

For MULTI-S01, encryption and decryption are tested. There are following types of tests for encryption:

- Variable Plaintext Known Answer Test (KAT-Text)
- Variable Key Known Answer Test (KAT-Key)
- Monte Carlo Test (MCT)
The test vectors for KAT are the same as those described in AESAVS[1] (refer to section 3.1) MCT is based on the algorithm described in AESAVS[1].

There are following types of test for decryption:

- Variable Ciphertext Known Answer Test (KAT-Text)
- Detection of Invalidity of Message Authentication Code for Altered Ciphertext

4.2.1 Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text)

The same KAT-Text as KAT-Text for AES with 128-bit key in ECB is performed. The default number of bits of plaintext(ciphertext) is defined in section 5.

4.2.2 Variable Key Known Answer Test (KAT-Key)

The same KAT-Key as KAT-Key for AES with 128-bit key in ECB with 128-bit plaintexts is performed.

4.2.3 Monte Carlo Test (MCT)

MCT is performed by inspecting whether the ciphertext $CT[i]$ computed by the following algorithm agrees with the expected value for every 128-bit initial plaintext, initial key, redundancy($R$), and initial vector(IV) randomly given. The default number of the iterations of loop innerloop outerloop are defined in section 5.

MCT for MULTI-S01

```plaintext
Key[0] = Key // 256-bit Initial Key
PT[0] = PT // 128-bit Initial plaintext
for (j=0; j<outerloop; j++)
{
    for (i=0; i<innerloop; i++ )
    {
        CT[i] = MULTI-S01(Key[i], PT[i], R, IV) // Encrypt 128-bit
        PT[i+1] = CT[i]
    }
    Output CT[i-1]
    Key[0] = Key[i-1] xor CT[i-1]
    PT[0] = CT[i-1]
}
```

4.3 128-bit RC4

For 128-bit RC4, encryption and decryption are tested.

There are following types of test for both encryption and decryption.

- Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text)
- Variable Key Known Answer Test (KAT-Key)
- Monte Carlo Test (MCT)

The test vectors for each KAT are the same as those described in AESAVS[1] (refer to section 3.1) MCT is based on the algorithm described in AESAVS[1].

4.3.1 Variable Plaintext (Ciphertext) Known Answer Test (KAT-Text)

The same KAT-Text as KAT-Text for AES with 128-bit key in ECB is performed. The default number of bits of plaintext(ciphertext) is defined in section 5.
4.3.2 Variable Key Known Answer Test (KAT-Key)

The same KAT-Key as KAT-Key for AES with 128-bit key in ECB with 128-bit plaintexts (ciphertexts) is performed.

4.3.3 Monte Carlo Test (MCT)

MCT is performed by inspecting whether the ciphertext(or plaintext) $CT[i]$ computed by the following algorithm agrees with the expected value for every 128-bit initial ciphertext(or plaintext) and initial key randomly given. The default numbers of the iterations of loop $innerloop$ and $outerloop$ are defined in section 5.

**MCT for 128-bit RC4**

```plaintext
Key[0] = Key // Initial Key(128-bit)
PT[0] = PT // Initial plaintext(128-bit)
for (j=0; j<outerloop; j++)
{
    for (i=0; i<innerloop; i++)
    {
        CT[i] = RC4(Key[i], PT[i]) // Encrypt (or decrypt) 128-bit
        Key[i+1] = key[i] xor CT[i]
        PT[i+1] = CT[i]
    }
    Output CT[i-1]
    Key[0] = Key[i-1] xor CT[i-1]
    PT[0] = CT[i-1]
}```
5 Conditions for Issuing Cryptographic Algorithm Validation Certificate

5.1 Details of Conditions

Requirements and default values of the parameters used for cryptographic algorithm implementation testing are shown in Table 5.1~5.4. In these tables, the first columns indicate the functions to be tested. There are two classes of functions. One is the class of mandatory functions for validation. The other is the class of optional functions. Mandatory functions are indicated by highlighted text in these tables.

For Symmetric-Key Cryptography, the conditions for issuing cryptographic algorithm validation certificate are:

- IUT shall implement at least one mandatory function.
- IUT shall pass the cryptographic algorithm implementation test.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Function</th>
<th>Parameter</th>
<th>Default</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Enc./Dec.</td>
<td>The number of MMT blocks</td>
<td>10</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCT inner loop iterations</td>
<td>1000</td>
<td>≥ 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCT outer loop iterations</td>
<td>100</td>
<td>≥ 100</td>
</tr>
</tbody>
</table>

Table 5.2: 3-key Triple DES

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Function</th>
<th>Parameter</th>
<th>Default</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-key Triple DES</td>
<td>Enc./Dec.</td>
<td>The number of MMT blocks</td>
<td>10</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCT inner loop iterations</td>
<td>10000</td>
<td>≥ 10000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCT outer loop iterations</td>
<td>400</td>
<td>≥ 400</td>
</tr>
</tbody>
</table>

Table 5.3: Other block ciphers

<table>
<thead>
<tr>
<th>The length of block</th>
<th>Function</th>
<th>Parameter</th>
<th>Default</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit</td>
<td>Enc./Dec.</td>
<td>The number of MMT blocks</td>
<td>10</td>
<td>≥ 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCT inner loop iterations</td>
<td>1000</td>
<td>≥ 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCT outer loop iterations</td>
<td>100</td>
<td>≥ 100</td>
</tr>
</tbody>
</table>

<p>| 128-bit             | Enc./Dec.| The number of MMT blocks      | 10      | ≥ 1        |
|                     |          | MCT inner loop iterations     | 1000    | ≥ 1000     |
|                     |          | MCT outer loop iterations     | 100     | ≥ 100      |</p>
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Function</th>
<th>Parameter</th>
<th>Default</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUGI</td>
<td>PRNG</td>
<td>The number of units</td>
<td>10</td>
<td>≤ 250</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inner loop iterations</td>
<td>1000</td>
<td>≥ 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outer loop iterations</td>
<td>100</td>
<td>≥ 100</td>
</tr>
<tr>
<td>MULTI-S01</td>
<td>Enc.</td>
<td>The bit length of plaintext</td>
<td>256</td>
<td>Multiple of 64 and ≤ 16000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inner loop iterations</td>
<td>1000</td>
<td>≥ 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outer loop iterations</td>
<td>100</td>
<td>≥ 100</td>
</tr>
<tr>
<td></td>
<td>Dec.</td>
<td>The bit length of ciphertext</td>
<td>256</td>
<td>Multiple of 64 and 192 ≤ x ≤ 16000</td>
</tr>
<tr>
<td>RC4</td>
<td>Enc.</td>
<td>The bit length of plaintext</td>
<td>128</td>
<td>Multiple of 8 and ≤ 16000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inner loop iterations</td>
<td>1000</td>
<td>≥ 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outer loop iterations</td>
<td>100</td>
<td>≥ 100</td>
</tr>
<tr>
<td></td>
<td>Dec.</td>
<td>The bit length of plaintext</td>
<td>128</td>
<td>Multiple of 8 and ≤ 16000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inner loop iterations</td>
<td>1000</td>
<td>≥ 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outer loop iterations</td>
<td>100</td>
<td>≥ 100</td>
</tr>
</tbody>
</table>
Supplementary Provision
This procedure shall come into force as of April 1, 2009, and shall be applicable as of April 1, 2009.

References


<table>
<thead>
<tr>
<th>Date of Revision</th>
<th>Prepared / Approved by</th>
<th>Revision Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 1, 2009</td>
<td>Hashimoto / Nakata</td>
<td>Newly Created</td>
</tr>
</tbody>
</table>