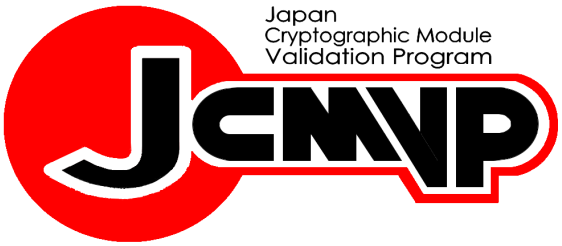


暗号アルゴリズム実装試験仕様書
– 共通鍵 –

令和元年 7 月 11 日

IPA

ATR-01-B

Cryptographic Algorithm Implementation Testing Requirements

独立行政法人情報処理推進機構

目 次

1 目的

本書は, 暗号アルゴリズム実装試験ツール (JCATT) に実装された共通鍵に関する暗号アルゴリズム実装試験仕様を記述するものである. 試験の対象とする暗号アルゴリズムは, 2 章に示す通りである.

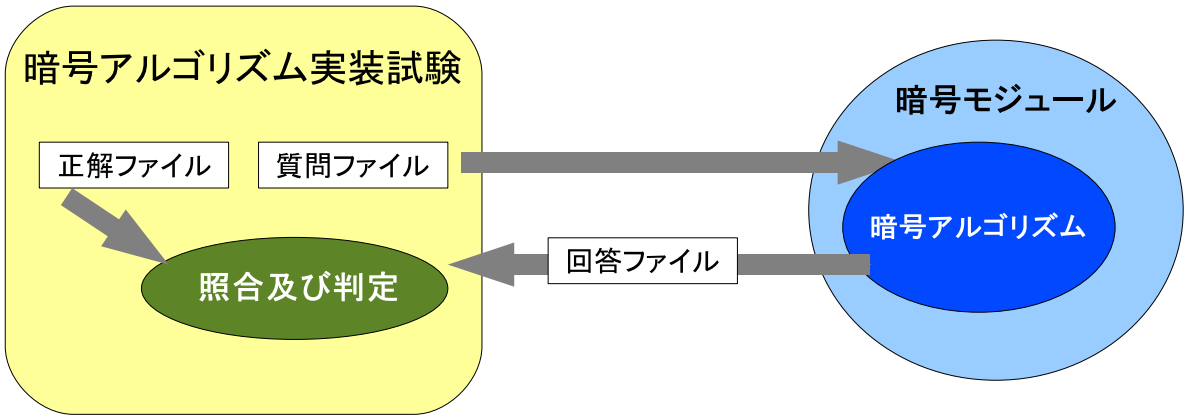
1.1 暗号アルゴリズム実装試験ツールの概要

暗号アルゴリズム実装試験ツールは次の特長を持つ.

- 試験対象の実装が暗号アルゴリズム仕様書に記述された事項に従って実装されているかどうかを試験する.
- 例えば共通鍵暗号の場合は暗号化, 復号など, 各暗号が有する機能ごとに試験を行う.
- 暗号アルゴリズム実装試験ツールと試験対象の実装は, 各種ファイルを介してデータの通信を行う. このことにより, 様々なプラットフォーム上の暗号実装を試験可能となる. ここで, ツールで使う各種ファイルの内訳は以下のとおりである.
  - 質問ファイル: 暗号アルゴリズム実装試験ツールが生成するファイル. 暗号アルゴリズムに対する入力データ及び制御情報が記録されている. 暗号モジュール試験機関からベンダ側へ送る.
  - 正解ファイル: 暗号アルゴリズム実装試験ツールが質問ファイルと同時に生成するファイル. 暗号アルゴリズムに対する入力データ, 制御情報及び対応する出力データが記録されている. 暗号モジュール試験機関で保存し, 回答ファイルが送られてきた際に回答ファイルと照合する.
  - 回答ファイル: ベンダ側で, 質問ファイルを元に暗号モジュールが生成したテキストファイル. ベンダから暗号モジュール試験機関側へ送る.

ファイルフォーマットは文献 [8], サンプルファイルは文献 [9] を参照.
暗号アルゴリズム実装試験の流れは, 図 1.1 の通りである.

図 1.1: 暗号アルゴリズム実装試験の流れ



1.2 本書の構成

本書の以降の構成は次の通りである.

- 2 章: 暗号アルゴリズム実装試験ツールが試験の対象とする暗号アルゴリズムを示す.

- 3 章以降: 各暗号の試験項目を記述する.

なお, 本書を通して次の略語を使用する.

- JCATT: 暗号アルゴリズム実装試験ツール
- IUT: JCATT が試験の対象とする実装

2　本書で対象とするセキュリティ機能

本書が試験対象とする暗号アルゴリズムを次に示す.

2.1　承認されたセキュリティ機能

2.1.1　ブロック暗号

64 ビットブロック
- 3-key Triple DES

128 ビットブロック
- AES
- Camellia

2.1.2　ストリーム暗号
- KCipher-2

2.2　暗号アルゴリズム確認対象非承認セキュリティ機能

2.2.1　ブロック暗号

64 ビットブロック
- CIPHERUNICORN-E
- Hierocrypt-L1
- MISTY1

128 ビットブロック
- CIPHERUNICORN-A
- Hierocrypt-3
- SC2000

2.2.2　ストリーム暗号
- MUGI
- MULTI-S01

3 暗号アルゴリズム実装試験仕様 – ブロック暗号 –

64 ビットブロック暗号アルゴリズム, 3-key Triple DES , CIPHERUNICORN-E , Hierocrypt-L1 , MISTY1 , ならびに 128 ビットブロック暗号アルゴリズム, AES, Camellia , CIPHERUNICORN -A , Hierocrypt-3 , SC2000 の試験対象機能および各試験対象機能に対する試験項目を記述する.

ブロック暗号の試験対象機能は次の通りである.

- ECB モード暗号化/復号機能
- CBC モード暗号化/復号機能
- CFB モード暗号化/復号機能
- OFB モード暗号化/復号機能
- CTR モード暗号化/復号機能

ここで, OFB モードと CFB モードのフィードバックビット幅はブロック幅とする. ただし, AES と 3-key Triple DES に対しては, 次の機能に対しても試験を実施することができる.

- CFB-1 モード暗号化/復号機能
- CFB-8 モード暗号化/復号機能

AES に対しては, 次の機能に対しても試験を実施することができる.

- XTS モード暗号化/復号機能

また, CTR モードは, NIST SP 800-38A の Appendix B.1[3] に記述されているインクリメンタルカウンタ, $x \leftarrow x+1 \bmod 2^m$, により実装されている場合を試験の対象とする.

128 ビットブロック暗号 (AES , Camellia , CIPHERUNICORN-A , Hierocrypt-3 , SC2000) の場合, 鍵長は 128, 192, 256 ビットの 3 種類である. したがって, 上記の各機能が 3 種類ずつに増えることに注意.

ブロック暗号の各機能に対する試験項目を次に示す.

AES

- 種々の平文 (暗号文) に対する既知入出力試験 (KAT-Text)
- 種々の鍵に対する既知入出力試験 (KAT-key)
- マルチブロックメッセージ試験 (MMT)
- モンテカルロ試験 (MCT)
- GFSbox 既知入出力試験 (KAT-GFSbox)
- KeySbox 既知入出力試験 (KAT-KeySbox)

3-key Triple DES

- 種々の平文 (暗号文) に対する既知入出力試験 (KAT-Text)
- 種々の鍵に対する既知入出力試験 (KAT-Key)
- マルチブロックメッセージ試験 (MMT)
- モンテカルロ試験 (MCT)
- 逆転置既知入出力試験 (KAT-IP)
- 置換既知入出力試験 (KAT-PO)
- Sbox 既知入出力試験 (KAT-ST)

他のブロック暗号

- 種々の平文 (暗号文) に対する既知入出力試験 (KAT-Text)
- 種々の鍵に対する既知入出力試験 (KAT-Key)
- マルチブロックメッセージ試験 (MMT)
- モンテカルロ試験 (MCT)
- Sbox 既知入出力試験 (KAT-Sbox)

AES の試験項目は AESAVS[1], 3-key Triple DES の試験項目は TMOVS[2] に準拠する. ただし, 各検定項目のパラメータ (MCT のループ数など) は別に定める規定値とする. また, AES と 3-key Triple DES に対する各既知入出力試験用のテストベクタは, それぞれ AESAVS および TMOVS に記述されているものを使用する.

AES と 3-key Triple DES 以外の暗号モジュールに対する試験項目は, AESAVS[1] に準拠する.

ブロック暗号の試験項目の詳細を以下に記述する.

3.1 種々の平文 (暗号文) に対する既知入出力試験 (KAT-Text)

KAT-Text では, 平文 (または暗号文) を様々に変化させ, 暗号文 (または平文) が期待値と一致するかどうかを試験する.

暗号化機能に対する入力平文 (または IV, カウンタ) には, 次のように 1 ビットずつ変化させたデータを用いる.

AESAVS 記載の KAT-Text 用入力平文 (または IV, カウンタ):

```
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
...
0xffffffff fffffffe fffffffe fffffffe
0xffffffff ffffffff ffffffff ffffffff
```

TMOVS 記載の KAT-Text 用入力平文 (または IV, カウンタ):

```
0x80000000 00000000
0x40000000 00000000
0x20000000 00000000
0x10000000 00000000
0x08000000 00000000
0x04000000 00000000
```

0x02000000 00000000
0x01000000 00000000
0x00800000 00000000
0x00400000 00000000
0x00200000 00000000
0x00100000 00000000
...
0x00000000 00000002
0x00000000 00000001

各モードにおけるテストベクタを表 3.1, 3.2, 3.3 に示す. CTR モードは, OFB モードと同様とした. また, AESAVS において復号機能に対する KAT-Text では, 入力暗号文として用いる暗号文は, KAT-Text 暗号化の際の期待値暗号文である.

表 3.1: AESAVS のテストベクタ：KAT-Text 暗号化

モード	平文	IV またはカウンタ	鍵
ECB	上記テストベクタ	—	全ゼロ
CBC	上記テストベクタ	全ゼロ	全ゼロ
CFB128	全ゼロ	上記テストベクタ	全ゼロ
CFB1	全ゼロ	上記テストベクタ	全ゼロ
CFB8	全ゼロ	上記テストベクタ	全ゼロ
OFB	全ゼロ	上記テストベクタ	全ゼロ
CTR	全ゼロ	上記テストベクタ	全ゼロ

表 3.2: TMOVS テストベクタ：KAT-Text 暗号化

モード	平文	IV またはカウンタ	鍵
ECB	上記テストベクタ	—	0101 ··· 01(奇数パリティ)
CBC	上記テストベクタ	全ゼロ	0101 ··· 01(奇数パリティ)
CFB64	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
CFB1	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
CFB8	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
OFB	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
CTR	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)

表 3.3: TMOVS テストベクタ：KAT-Text 復号

モード	暗号文	IV またはカウンタ	鍵
ECB	KAT-Text 暗号化の際の暗号文	—	0101 ··· 01(奇数パリティ)
CBC	KAT-Text 暗号化の際の暗号文	全ゼロ	0101 ··· 01(奇数パリティ)
CFB64	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
CFB1	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
CFB8	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
OFB	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)
CTR	全ゼロ	上記テストベクタ	0101 ··· 01(奇数パリティ)

3.2 種々の鍵に対する既知入出力試験 (KAT-Key)

KAT-Key では, 鍵を様々に変化させ, 暗号文 (または平文) が期待値と一致するかどうかを試験する.

暗号化機能に対する鍵には, 次のように 1 ビットずつ変化させたデータを用いる.

AESAVS 記載の KAT-Key 用の入力鍵:

```
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
...
0xffffffff ffffffff ffffffff fffffffe
0xffffffff ffffffff ffffffff ffffffff
```

TMOVS KAT-Key の鍵:

```
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
```

各モードにおけるテストベクタを表 3.4, 3.5, 3.6 に示す. CTR モードは, OFB モードと同様とした. また, AESAVS において復号機能に対する KAT-Key では, 入力暗号文として用いる暗号文は, KAT-Key 暗号化の際の期待値暗号文である.

3.3 マルチブロックメッセージ試験 (MMT)

MMT ではランダムに与えられた複数ブロック分の平文 (または暗号文), IV, カウンタに対する暗号文 (または平文) が期待値と一致するかどうかで検証を行う. ただし, MMT ブロック数は別に定

表 3.4: AESAVS テストベクタ：KAT-Key 暗号化

モード	平文	IV またはカウンタ	鍵
ECB	全ゼロ	—	上記テストベクタ
CBC	全ゼロ	全ゼロ	上記テストベクタ
CFB128	全ゼロ	全ゼロ	上記テストベクタ
CFB1	全ゼロ	全ゼロ	上記テストベクタ
CFB8	全ゼロ	全ゼロ	上記テストベクタ
OFB	全ゼロ	全ゼロ	上記テストベクタ
CTR	全ゼロ	全ゼロ	上記テストベクタ

表 3.5: TMOVS テストベクタ：KAT-Key 暗号化

モード	平文	IV またはカウンタ	鍵
ECB	全ゼロ	—	上記テストベクタ
CBC	全ゼロ	全ゼロ	上記テストベクタ
CFB64	全ゼロ	全ゼロ	上記テストベクタ
CFB1	全ゼロ	全ゼロ	上記テストベクタ
CFB8	全ゼロ	全ゼロ	上記テストベクタ
OFB	全ゼロ	全ゼロ	上記テストベクタ
CTR	全ゼロ	全ゼロ	上記テストベクタ

表 3.6: TMOVS テストベクタ：KAT-Key 復号

モード	暗号文	IV またはカウンタ	鍵
ECB	KAT-Key 暗号化の際の暗号文	—	上記テストベクタ
CBC	KAT-Key 暗号化の際の暗号文	全ゼロ	上記テストベクタ
CFB64	全ゼロ	全ゼロ	上記テストベクタ
CFB1	全ゼロ	全ゼロ	上記テストベクタ
CFB8	全ゼロ	全ゼロ	上記テストベクタ
OFB	全ゼロ	全ゼロ	上記テストベクタ
CTR	全ゼロ	全ゼロ	上記テストベクタ

める規定値とする.

3.4 モンテカルロ試験 (MCT)

MCT では, ランダムに与えられた 1 ブロック分の初期平文 (または暗号文), 初期鍵, 初期 IV に対して, 次のアルゴリズムで計算される暗号文 (または平文)`CT[i]` が期待値と一致するかどうかで検証を行う. ただし, 内側ループ回数 `innerloop` および外側ループ回数 `outerloop` は別途定める規定値とする.

3.4.1 3-key Triple DES に対する MCT

3-key Triple DES に対する MCT を以下に記述する. TMOVS に記述されているアルゴリズムと同じである. ただし, CTR モード (インクリメンタルカウンタ) は TMOVS に記述されていないため, ECB ードに対する MCT に準拠した. なお, TMOVS に記述されているループ回数の規定値は, innerloop=10,000, outerloop=400 である.

3.4.1.1 ECB モード

3.4.1.1.1 ECB モード暗号化

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
PT = PT_0      // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], PT) // ECB モード暗号化
    PT = CT[j]
  }
  Output CT[innerloop-1] // 内側ループ内で計算された最後の暗号文
  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]
}
```

3.4.1.1.2 ECB モード復号

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
CT = CT_0      // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key1[i], Key2[i], Key3[i], CT) // ECB モード復号
    CT = PT[j]
  }
  Output PT[innerloop-1] // 内側ループ内で計算された最後の平文
  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 CBC モード

3.4.1.2.1 CBC モード暗号化

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV
PT = PT_0      // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // CBC モード暗号化
    If( i==0 )
```

```
    {
      PT = IV
    }
    else
    {
      PT = CT[j-1]
    }
    IV = CT[j]
  }
  Output CT[innerloop-1] // 内側ループ内で計算された最後の暗号文
  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 モード復号

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV
CT = CT_0      // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key1[i], Key2[i], Key3[i], IV, CT) // CBC モード復号
    IV = CT
    CT = PT[j]
  }
  Output PT[innerloop-1] // 内側ループ内で計算された最後の平文
  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]
}
```

3.4.1.3 CFB モード

3.4.1.3.1 CFB モード暗号化

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV(64 ビット)
PT = PT_0      // 初期平文 (64 ビット)
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // CFB モード暗号化
    PT = IV
    IV = CT[j]
  }
  Output CT[innerloop-1] // 内側ループ内で計算された最後の暗号文
  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 モード復号

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV(64 ビット)
CT = CT_0      // 初期暗号文 (64 ビット)
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key1[i], Key2[i], Key3[i], IV, CT) // CFB モード復号
    IV = CT
    CT = O[j] // O[j]=Decryption() 内で暗号文を xor する直前の値
  }
  Output PT[innerloop-1] // 内側ループ内で計算された最後の平文
  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 CFB-1 モード

3.4.1.4.1 CFB-1 モード暗号化

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV(64 ビット)
PT = PT_0      // 初期平文 (1 ビット)
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], IV, PT) // CFB-1 モード暗号化
    PT = IV の左 1 ビット
    IV = IV の右 63 ビット||CT[j]
  }
  Output CT[innerloop-1] // 内側ループ内で計算された最後の暗号文
  // 暗号文を連結して 192 ビットにする
  C = CT[innerloop-192]||...||CT[innerloop-2]||CT[innerloop-1]
  Key1[i+1] = Key1[i] xor "C の右 64 ビット"
  Key2[i+1] = Key2[i] xor "C の中 64 ビット"
  Key3[i+1] = Key3[i] xor "C の左 64 ビット"
}
```

3.4.1.4.2 CFB-1 モード復号

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV(64 ビット)
CT = CT_0      // 初期暗号文 (1 ビット)
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 モード復号
    IV = IV の右 63 ビット||CT
    CT = O[j] の左 1 ビット // O[j]=Decryption() 内で暗号文を xor する直前の値
  }
  Output PT[innerloop-1] // 内側ループ内で計算された最後の平文
  // 平文を連結して 192 ビットにする
  P = PT[innerloop-192]||...||PT[innerloop-2]||PT[innerloop-1]
  Key1[i+1] = Key1[i] xor "P の右 64 ビット"
  Key2[i+1] = Key2[i] xor "P の中 64 ビット"
  Key3[i+1] = Key3[i] xor "P の左 64 ビット"
}
```

3.4.1.5 CFB-8 モード

3.4.1.5.1 CFB-8 モード暗号化

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV(64 ビット)
PT = PT_0      // 初期平文 (8 ビット)
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 モード暗号化
    PT = IV の左 8 ビット
    IV = IV の右 56 ビット||CT[j]
  }
  Output CT[innerloop-1] // 内側ループ内で計算された最後の暗号文
  // 暗号文を連結して 192 ビットにする
  C = CT[innerloop-24]||...||CT[innerloop-2]||CT[innerloop-1]
  Key1[i+1] = Key1[i] xor "C の右 64 ビット"
  Key2[i+1] = Key2[i] xor "C の中 64 ビット"
  Key3[i+1] = Key3[i] xor "C の左 64 ビット"
}
```

3.4.1.5.2 CFB-8 モード復号

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV(64 ビット)
CT = CT_0      // 初期暗号文 (8 ビット)
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 モード復号
    IV = IV の右 56 ビット||CT
    CT = O[j] の左 8 ビット // O[j]=Decryption() 内で暗号文を xor する直前の値
  }
```

```
  Output PT[innerloop-1] // 内側ループ内で計算された最後の平文
  // 平文を連結して 192 ビットにする
  P = PT[innerloop-24]|||...|PT[innerloop-2]||PT[innerloop-1]
  Key1[i+1] = Key1[i] xor "P の右 64 ビット"
  Key2[i+1] = Key2[i] xor "P の中 64 ビット"
  Key3[i+1] = Key3[i] xor "P の左 64 ビット"
}
```

3.4.1.6 OFB モード

3.4.1.6.1 OFB モード暗号化

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV
PT = PT_0      // 初期平文
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 モード暗号化
    PT = IV
    IV = O[j] // O[j]=Encryption() 内で平文を xor する直前の値
  }
  Output CT[innerloop-1] // 内側ループ内で計算された最後の暗号文
  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 モード復号

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
IV = IV_0      // 初期 IV
CT = CT_0      // 初期暗号文
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 モード復号
    CT = IV
    IV = O[j] // O[j]=Decryption() 内で暗号文を xor する直前の値
  }
  Output PT[innerloop-1] // 内側ループ内で計算された最後の平文
  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 CTR モード

3.4.1.7.1 CTR モード暗号化

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
CTR = CTR_0    // 初期カウンタ
PT = PT_0      // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key1[i], Key2[i], Key3[i], CTR, PT) // CTR モード暗号化
    CTR = (CTR + 1) mod 2^64
    PT = CT[j]
  }
  Output CT[innerloop-1] // 内側ループ内で計算された最後の暗号文
  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 モード復号

```
Key1[0] = Key1 // 初期鍵 (64 ビット)
Key2[0] = Key2 // 初期鍵 (64 ビット)
Key3[0] = Key3 // 初期鍵 (64 ビット)
CTR = CTR_0    // 初期カウンタ
CT = CT_0      // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key1[i], Key2[i], Key3[i], CTR, CT) // CTR モード復号
    CTR = (CTR + 1) mod 2^64
    CT = PT[j]
  }
  Output PT[innerloop-1] // 内側ループ内で計算された最後の平文
  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 64 ビットブロック暗号 (3-key Triple DES を除く) に対する MCT

3.4.2.1 ECB モード

3.4.2.1.1 ECB モード暗号化

```
Key[0] = Key // 初期鍵
PT = PT_0    // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
```

```
  {
    CT[j] = Encryption(Key[i], PT) // ECB モード暗号化
    PT = CT[j]
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}
```

3.4.2.1.2 ECB モード復号

```
Key[0] = Key // 初期鍵
CT = CT_0    // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], CT) // ECB モード復号
    CT = PT[j]
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
}
```

3.4.2.2 CBC モード

3.4.2.2.1 CBC モード暗号化

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
PT = PT_0    // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // CBC モード暗号化
    If( j==0 )
    {
      PT = IV
    }
    else
    {
      PT = CT[j-1]
    }
    IV = CT[j]
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}
```

3.4.2.2.2 CBC モード復号

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
```

```
CT = CT_0    // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], IV, CT) // CBC モード復号
    If( j==0 )
    {
      tmp = CT
      CT = IV
      IV = tmp
    }
    else
    {
      IV = CT
      CT = PT[j-1]
    }
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  IV = PT[innerloop-1]
}
```

3.4.2.3 CFB モード

3.4.2.3.1 CFB モード暗号化

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
PT = PT_0    // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // CFB モード暗号化
    If( j==0 )
    {
      PT = IV
    }
    else
    {
      PT = CT[j-1]
    }
    IV = CT[j]
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}
```

3.4.2.3.2 CFB モード復号

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
CT = CT_0    // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
```

```
    PT[j] = Decryption(Key[i], IV, CT) // CFB モード復号
    If( j==0 )
    {
      tmp = IV
      IV = CT
      CT = tmp
    }
    else
    {
      IV = CT
      CT = PT[j-1]
    }
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  IV = PT[innerloop-1]
}
```

3.4.2.4 OFB モード

3.4.2.4.1 OFB モード暗号化

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
PT = PT_0    // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // OFB モード暗号化
    If( j==0 )
    {
      PT = IV
    }
    else
    {
      PT = CT[j-1]
    }
    IV = O[j] // O[j]=Encryption() 内で平文を xor する直前の値
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
  IV = CT[innerloop-1]
}
```

3.4.2.4.2 OFB モード復号

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
CT = CT_0    // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], IV, CT) // OFB モード復号
    If( j==0 )
    {
      CT = IV
    }
```

```
    else
    {
      CT = PT[j-1]
    }
    IV = O[j] // O[j]=Decryption() 内で暗号文を xor する直前の値
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  IV = PT[innerloop-1]
}
```

3.4.2.5 CTR モード

3.4.2.5.1 CTR モード暗号化

```
Key[0] = Key    // 初期鍵
CTR = CTR_0     // 初期カウンタ
PT = PT_0       // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], CTR, PT) // CTR モード暗号化
    CRT = (CTR + 1) mod 2^64
    PT = CT[j]
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
}
```

3.4.2.5.2 CTR モード復号

```
Key[0] = Key    // 初期鍵
CTR = CTR_0     // 初期カウンタ
CT = CT_0       // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], CTR, CT) // CTR モード復号
    CRT = (CTR + 1) mod 2^64
    CT = PT[j]
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
}
```

3.4.3 128 ビットブロック暗号に対する MCT

3.4.3.1 ECB モード

3.4.3.1.1 ECB モード暗号化

```
Key[0] = Key // 初期鍵
PT = PT_0    // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], PT) // ECB モード暗号化
    PT = CT[j]
  }
  Output CT[innerloop-1]
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
  }
}
```

3.4.3.1.2 ECB モード復号

```
Key[0] = Key // 初期鍵
CT = CT_0    // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], CT) // ECB モード復号
    CT = PT[j]
  }
  Output PT[innerloop-1]
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  }
}
```

3.4.3.2 CBC モード

3.4.3.2.1 CBC モード暗号化

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
PT = PT_0    // 初期平文
for (i=0; i<outerloop; i++)
```

```
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // CBC モード暗号化
    If( j==0 )
    {
      PT = IV
    }
    else
    {
      PT = CT[j-1]
    }
    IV = CT[j]
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
  }
}
```

3.4.3.2.2 CBC モード復号

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
CT = CT_0    // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], IV, CT) // CBC モード復号
    If( j==0 )
    {
      tmp = CT
      CT = IV
      IV = tmp
    }
    else
    {
      IV = CT
      CT = PT[j-1]
    }
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  }
```

```
  IV = PT[innerloop-1] // 内側ループで計算された最後の平文
}
```

3.4.3.3 CFB モード

3.4.3.3.1 CFB モード暗号化

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV(128 ビット)
PT = PT_0    // 初期平文 (128 ビット)
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // CFB モード暗号化
    If( j==0 )
    {
      PT = IV
    }
    else
    {
      PT = CT[j-1]
    }
    IV = CT[j]
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
  }
  IV = CT[innerloop-1]
}
```

3.4.3.3.2 CFB モード復号

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV(128 ビット)
CT = CT_0    // 初期暗号文 (128 ビット)
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], IV, CT) // CFB モード復号
    If( j==0 )
    {
      tmp = IV
      IV = CT
      CT = tmp
    }
    else
    {
```

```
      IV = CT
      CT = PT[j-1]
    }
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  }
  IV = PT[innerloop-1]
}
```

3.4.3.4 CFB-1 モード (AES のみ)

3.4.3.4.1 CFB-1 モード暗号化

```
Key[0] = Key // 初期鍵
IV[0] = IV_0 // 初期 IV(128 ビット)
PT = PT_0    // 初期平文 (1 ビット)
for (i=0; i<outerloop; i++)
{
  IV = IV[i]
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // CFB-1 モード暗号化
    IV = IV の右 127 ビット||CT[j]
    if( j<128 )
    {
      PT = IV[i] の左 j 番目 1 ビット
    }
    else
    {
      PT = CT[j-128]
    }
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  If ( 鍵長 == 128 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-128] || CT[innerloop-127] || ...
               || CT[innerloop-1]
  }
  If ( 鍵長 == 192 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-192] || CT[innerloop-191] || ...
               || CT[innerloop-1]
  }
  If ( 鍵長 == 256 ビット )
  {
    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 モード復号

```
Key[0] = Key // 初期鍵
IV[0] = IV_0 // 初期 IV(128 ビット)
CT = CT_0    // 初期暗号文 (1 ビット)
for (i=0; i<outerloop; i++)
{
  IV = IV[i]
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], IV, CT) // CFB-1 モード復号
    IV = IV の右 127 ビット||CT
    if( j<128 )
    {
      CT = IV[i] の左 j 番目 1 ビット
    }
    else
    {
      CT = PT[j-128]
    }
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  If ( 鍵長 == 128 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-128] || PT[innerloop-127] || ...
               || PT[innerloop-1]
  }
  If ( 鍵長 == 192 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-192] || PT[innerloop-191] || ...
               || PT[innerloop-1]
  }
  If ( 鍵長 == 256 ビット )
  {
    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 CFB-8 モード (AES のみ)

3.4.3.5.1 CFB-8 モード暗号化

```
Key[0] = Key // 初期鍵
IV[0] = IV_0 // 初期 IV(128 ビット)
PT = PT_0    // 初期平文 (8 ビット)
for (i=0; i<outerloop; i++)
{
  IV = IV[i]
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // CFB-8 モード暗号化
    IV = IV の右 120 ビット||CT[j]
    if( j<16 )
    {
      PT = IV[i] の左 j 番目 1 バイト
    }
    else
    {
      PT = CT[j-16]
    }
```

```
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  If ( 鍵長 == 128 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-16] || CT[innerloop-15] || ...
               || CT[innerloop-1]
  }
  If ( 鍵長 == 192 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-24] || CT[innerloop-23] || ...
               || CT[innerloop-1]
  }
  If ( 鍵長 == 256 ビット )
  {
    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 モード復号

```
Key[0] = Key // 初期鍵
IV[0] = IV_0 // 初期 IV(128 ビット)
CT = CT_0    // 初期暗号文 (8 ビット)
for (i=0; i<outerloop; i++)
{
  IV = IV[i]
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], IV, CT) // CFB-8 モード復号
    IV = IV の右 120 ビット||CT
    if( j<16 )
    {
      CT = IV[i] の左 j 番目 1 バイト
    }
    else
    {
      CT = PT[j-16]
    }
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-16] || PT[innerloop-15] || ...
               || PT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-24] || PT[innerloop-23] || ...
               || PT[innerloop-1]
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-32] || PT[innerloop-31] || ...
               || PT[innerloop-1]
  }
  IV[i+1] = PT[innerloop-16] || PT[innerloop-15] ||...|| PT[innerloop-1]
}
```

3.4.3.6 OFB モード

3.4.3.6.1 OFB モード暗号化

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
PT = PT_0    // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], IV, PT) // OFB モード暗号化
    If( j==0 )
    {
      PT = IV
    }
    else
    {
      PT = CT[j-1]
    }
    IV = O[j] // O[j]=Encryption() 内で平文を xor する直前の値
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
  }
  IV = CT[innerloop-1]
}
```

3.4.3.6.2 OFB モード復号

```
Key[0] = Key // 初期鍵
IV = IV_0    // 初期 IV
CT = CT_0    // 初期暗号文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], IV, CT) // OFB モード復号
    If( j==0 )
    {
      CT = IV
    }
    else
    {
      CT = PT[j-1]
    }
    IV = O[j] // O[j]=Decryption() 内で暗号文を xor する直前の値
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor PT[innerloop-1]
```

```
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  }
  IV = PT[innerloop-1]
}
```

3.4.3.7 CTR モード

3.4.3.7.1 CTR モード暗号化

```
Key[0] = Key    // 初期鍵
CTR = CTR_0     // 初期カウンタ
PT = PT_0       // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    CT[j] = Encryption(Key[i], CTR, PT) // CTR モード暗号化
    CTR = (CTR + 1) mod 2^128
    PT = CT[j]
  }
  Output CT[innerloop-1] // 内側ループで計算された最後の暗号文
  If ( 鍵長==128 ビット )
  {
    Key[i+1] = Key[i] xor CT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of CT[innerloop-2] || CT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (CT[innerloop-2] || CT[innerloop-1])
  }
}
```

3.4.3.7.2 CTR モード復号

```
Key[0] = Key    // 初期鍵
CTR = CTR_0     // 初期カウンタ
CT = CT_0       // 初期平文
for (i=0; i<outerloop; i++)
{
  for (j=0; j<innerloop; j++)
  {
    PT[j] = Decryption(Key[i], CTR, CT) // CTR モード復号
    CTR = (CTR + 1) mod 2^128
    CT = PT[j]
  }
  Output PT[innerloop-1] // 内側ループで計算された最後の平文
  If ( 鍵長==128 ビット )
```

```
  {
    Key[i+1] = Key[i] xor PT[innerloop-1]
  }
  If ( 鍵長==192 ビット )
  {
    Key[i+1] = Key[i] xor (last 64-bits of PT[innerloop-2] || PT[innerloop-1])
  }
  If ( 鍵長==256 ビット )
  {
    Key[i+1] = Key[i] xor (PT[innerloop-2] || PT[innerloop-1])
  }
}
```

3.5 Sbox 既知入出力試験 (KAT-Sbox)

KAT-Sbox は, 各暗号アルゴリズムの Sbox テーブルのエントリーを全て 1 回以上引くように作成された入力データ (平文, 暗号文, 鍵, IV, カウンタ) を用いる検証方法である. 入力データは, 各ブロック暗号ごとに作成されたものを用いる.

AES に対する KAT-GFSbox と KAT-KeySbox の入力データは, AESAVS[1] に記載の入力データを用いる.

3-key Triple DES に対する KAT-ST の入力データは, TMOVS[2] に記載の入力データを用いる.

3.6 3-key Triple DES に対するその他の KAT

3-key Triple DES に対する次の KAT の入力データは, TMOVS[2] に記載の入力データを用いる.

- 逆転置既知入出力試験 (KAT-IP)
- 置換既知入出力試験 (KAT-PO)

3.7 XTS モード

NIST SP800-38E[4] の参照先である IEEE1619[5] で規定されている XTS モードに対しては, XTSVS[6] に記述された試験を行う. 試験対象機能は次の通りである.

- 暗号化機能
- 復号機能

XTS モードは, 承認された 128 ビットブロック暗号アルゴリズムと組み合わせて使用する. なお, XTS モードの試験に先立って, ECB モードでの暗号アルゴリズム実装試験に合格している必要がある.

3.7.1 暗号化機能試験

暗号化機能試験では, 入力データ (平文, 暗号鍵及び tweak) を様々に変化させ, 暗号文が期待値と一致するかどうかを試験する. 入力データの組の数は, 別途定める規定値とする.

3.7.2 復号機能試験

復号機能試験では, 入力データ (暗号文, 暗号鍵及び tweak) を様々に変化させ, 平文が期待値と一致するかどうかを試験する. 入力データの組の数は, 別途定める規定値とする.

3.7.3 データユニット長に関する要件

XTS モードのデータユニット長は 2^{20} ブロック以下であることが NIST SP800-38E[4] において要求されている. この要求事項は JCATT では試験できないため, IUT が要件を満たす証拠をベンダが提供しなければならない. 試験者は, ドキュメント, ソースコードのレビューによってこの要求事項を満たしていることを確認する.

4 暗号アルゴリズム実装試験仕様 – ストリーム暗号 –

ストリーム暗号アルゴリズム, KCipher-2 の暗号モジュール試験項目を記述する.

4.1 KCipher-2

KCipher-2 の試験対象機能は次の 1 つである.

- 擬似乱数生成機能

擬似乱数生成機能試験の試験項目は次の通りである.

- 種々の鍵に対する既知入出力試験 (KAT-Key)
- 種々の IV に対する既知入出力試験 (KAT-IV)
- Table 参照試験 (KAT-Table)
- モンテカルロ試験 (MCT)

各試験項目の詳細を以下に記述する.

4.1.1 種々の鍵に対する既知入出力試験 (KAT-Key)

初期鍵, IV に対する複数ブロック分の鍵系列が期待値と一致するかどうかで検証を行う. ただし, ブロック数は別に定める規定値とする.

初期鍵には, 次のように 1 ビットずつ変化させたデータを用いる.

KAT-Key 用の入力初期鍵:

```
0x00000000 00000000 00000000 00000000
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
...
0xffffffff fffffffe fffffffe fffffffe
0xffffffff ffffffff ffffffff ffffffff
```

テストベクタを表 4.1 に示す.

表 4.1: KCipher-2 のテストベクタ：KAT-Key

IV	初期鍵
全ゼロ	上記テストベクタ

4.1.2 種々の IV に対する既知入出力試験 (KAT-IV)

初期鍵, IV に対する複数ブロック分の鍵系列が期待値と一致するかどうかで検証を行う. ただし, ブロック数は別に定める規定値とする.

IV には, 次のように 1 ビットずつ変化させたデータを用いる.

KAT-IV 用の入力 IV:

```
0x00000000 00000000 00000000 00000000
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
...
0xffffffff ffffffff ffffffff fffffffe
0xffffffff ffffffff ffffffff ffffffff
```

テストベクタを表 4.2 に示す.

表 4.2: KCipher-2 のテストベクタ：KAT-IV

IV	初期鍵
上記テストベクタ	全ゼロ

4.1.3 Table 参照試験 (KAT-Table)

KAT-Table は, KCipher-2 のテーブルのエントリーを全て 1 回以上引くように作成された入力データ (初期鍵, IV) を用いる検証方法である. テーブルのエントリーを全て 1 回以上引くような試験条件の選び方として, KCipher-2 の仕様書 [7] に示された参照テーブルを使う試験対象実装を想定し, テーブルのエントリーを全て 1 回以上引くようなブロック数分を試験対象実装に生成させることとする.

4.1.4 モンテカルロ試験 (MCT)

MCT では, ランダムに与えられた 64 ビットの初期平文 PT, 初期鍵 Key, 初期ベクタ IV に対して, 次のアルゴリズムで計算される暗号文 CT_j が期待値と一致するかどうかで検証を行う. ループ回数

innerloop および *outerloop* は別途定める規定値とする.

4.1.4.1 記号の定義

擬似コード上の表記	型	説明
i	整数	0 から $innerloop-1$ までの値をとる整数.
$innerloop$	整数	内側ループ回数
IV	ビット列	128 ビットの入力 IV.
IV_i	ビット列	内側ループのループカウンタが i のときに使用される入力 IV.
j	整数	0 から $outerloop-1$ までの値をとる整数.
Key	ビット列	128 ビットの入力初期鍵.
N_b	整数	内側ループの中で生成する鍵系列のブロック数.
N_c	整数	回答ファイルに記録する鍵系列のブロック数.
$outerloop$	整数	外側ループ回数
X, Y	ビット列	出力された鍵系列を格納する一時変数.

注
1. 網掛け ( ) された項目については, 質問ファイルで提供される.

4.1.4.2 関数定義

擬似コード上の表記	説明
KCipher-2(Key, IV, n)	KCipher-2 の鍵系列出力処理. 入力鍵 (Key) 及び入力 IV(IV) を用いて初期化処理を行い, 64 ビットを 1 ブロックとして鍵系列を n ブロック分出力する.
leftmost(V, a)	ビット列 V の左から a ビット
Output(V)	V を回答ファイルに出力する.
rightmost(V, a)	ビット列 V の右から a ビット
$U \oplus V$	同じ長さのビット列 U と V との排他的論理和

4.1.4.3 モンテカルロ試験 (MCT) のアルゴリズム

Algorithm 1 MCT Algorithm for KCipher-2

1: $IV_0 = IV$ ▷ 128 ビット分
2: **for** $j = 0$ to $outerloop - 1$ **do**
3: **for** $i = 0$ to $innerloop - 1$ **do**
4: $X = \textbf{KCipher-2}(Key, IV_i, N_b)$
▷ Key 及び IV_j を用いた初期化処理を行い, N_b ブロック分の鍵系列の出力を行う
5: $Y = \textbf{rightmost}(X, 64 \times N_c)$ ▷ 最終の N_c ブロック分のみを取り出す.
6: $IV_{i+1} = IV_i \oplus \textbf{rightmost}(Y, 128)$
7: **end for**
8: **Output**(Y)
▷ 内側ループで計算された鍵系列の, 最終の N_c ブロックを回答ファイルに記録する
9: $Key = Key \oplus (\textbf{leftmost}(Y, 128))$ ▷ 128 ビット分
10: $IV_0 = IV_{inner\_loop} \oplus (\textbf{rightmost}(Y, 128))$ ▷ 128 ビット分
11: **end for**

4.2 MUGI

MUGI の試験対象機能は,

- 擬似乱数生成機能

のみである. 試験項目は次の通りである.

- 種々の鍵に対する既知入出力試験 (KAT-Key)
- モンテカルロ試験 (MCT)

KAT のテストベクトルは, AESAVS に記載されているものに準拠する. また, MCT は, AESAVS に記述されたアルゴリズムに準拠する. 詳細を以下に記述する.

4.2.1 種々の鍵に対する既知入出力試験 (KAT-Key)

128 ビット鍵 ECB モード AES に対する KAT-Key と同じ KAT-Key を行う. ユニット数は別途定める規定値とする.

4.2.2 モンテカルロ試験 (MCT)

MCT では, ランダムに与えられた初期鍵および初期ベクトルに対して, 次のアルゴリズムで計算される乱数 CT_i が期待値と一致するかどうかで検証を行う. ループ回数 *innerloop* および *outerloop* は別途定める規定値とする.

4.2.2.1 記号の定義

擬似コード上の表記	型	説明
CT_i	ビット列	内側ループのループカウンタが i のときに出力される, n ユニット分の鍵系列.
i	整数	0 から $innerloop-1$ までの値をとる整数.
I_i	ビット列	内側ループのループカウンタが i のときに使用される入力 IV.
$innerloop$	整数	内側ループ回数
IV	ビット列	128 ビットの入力 IV.
j	整数	0 から $outerloop-1$ までの値をとる整数.
Key	ビット列	128 ビットの入力初期鍵.
Key_j	ビット列	外側ループのループカウンタが j のときに使用される入力鍵.
n	整数	ユニット数. 4.2.2.3 においては, 2 に固定.
$outerloop$	整数	外側ループ回数.

注
1. 網掛け ( ) された項目については, 質問ファイルで提供される.

4.2.2.2 関数定義

擬似コード上の表記	説明
MUGI(Key, IV, n)	MUGI の鍵系列出力処理. 入力鍵 (Key) 及び入力 IV(IV) を用いて初期化処理を行い, 64 ビットを 1 ユニットとして鍵系列を n ユニット分出力する.
Output(V)	V を回答ファイルに出力する.
$U \oplus V$	同じ長さのビット列 U と V との排他的論理和

4.2.2.3 モンテカルロ試験 (MCT) のアルゴリズム

Algorithm 2 MCT Algorithm for MUGI

1: $Key_0 = Key$ ▷ 初期鍵 128 ビット
2: $I_0 = IV$ ▷ 初期ベクトル 128 ビット
3: $n = 2$ ▷ ユニット数
4: **for** $j = 0$ to $outerloop - 1$ **do**
5: **for** $i = 0$ to $innerloop - 1$ **do**
6: $CT_i = \mathbf{MUGI}(Key_j, I_i, n)$
7: $I_{i+1} = CT_i$
8: **end for**
9: **Output**$(CT_{innerloop-1})$
▷ 内側ループで計算された鍵系列の, 最終の n(=2) ユニットを回答ファイルに記録する
10: $Key_{j+1} = Key_j \oplus CT_{innerloop-1}$ ▷ 128 ビット分
11: $I_0 = CT_{innerloop-1}$ ▷ 128 ビット分
12: **end for**

4.3 MULTI-S01

MULTI-S01 の試験対象機能は次の 2 つである.

- 暗号化機能
- 復号機能

暗号化機能試験の試験項目は次の通りである.

- 種々の平文に対する既知入出力試験 (KAT-Text)
- 種々の鍵に対する既知入出力試験 (KAT-Key)
- モンテカルロ試験 (MCT)

各 KAT のテストベクトルは, AESAVS に記載されているものと同様とする (3.1 節参照). また, MCT は, AESAVS に記述されたアルゴリズムに準拠する.

復号機能試験の試験項目は次の通りである.

- 種々の暗号文に対する既知入出力試験 (KAT-Text)
- 改竄された暗号文に対してメッセージ認証子不正を検出すること.

各試験項目の詳細を以下に記述する.

4.3.1 種々の平文 (暗号文) に対する既知入出力試験 (KAT-Text)

128 ビット鍵 ECB モード AES に対する KAT-Text と同様な KAT-Text を行う. また, 入力平文 (暗号文) のビット数は別途定める規定値とする.

4.3.2 種々の鍵に対する既知入出力試験 (KAT-Key)

入力平文を 128 ビットとし, 128 ビット鍵 ECB モード AES に対する KAT-Key と同様な KAT-Key を行う.

4.3.3 モンテカルロ試験 (MCT)

MCT では, ランダムに与えられた 128 ビットの初期平文, 初期鍵, 冗長性 (R), 初期値 (IV) に対して, 次のアルゴリズムで計算される暗号文 CT_i が期待値と一致するかどうかで検証を行う. ループ回数 *innerloop* および *outerloop* は別途定める規定値とする.

4.3.3.1 記号の定義

擬似コード上の表記	型	説明
CT_i	ビット列	内側ループのループカウンタが i のときの出力暗号文.
i	整数	0 から $innerloop-1$ までの値をとる整数.
$innerloop$	整数	内側ループ回数
IV	ビット列	256 ビットの入力 IV.
j	整数	0 から $outerloop-1$ までの値をとる整数.
Key	ビット列	128 ビットの入力初期鍵.
Key_j	ビット列	外側ループのループカウンタが j のときに使用される入力鍵.
$outerloop$	整数	外側ループ回数.
PT	ビット列	128 ビットの初期平文.
PT_i	ビット列	内側ループのループカウンタが i のときに使用される入力平文.
R	ビット列	冗長性 (64 ビット).

注
1. 網掛け ( ) された項目については, 質問ファイルで提供される.

4.3.3.2 関数定義

擬似コード上の表記	説明
leftmost(V,a)	ビット列 V の左から a ビット
MULTI-S01(Key,M,R,Q)	MULTI-S01 の暗号化処理. 入力鍵 (Key), メッセージ (M), 冗長性 (R) 及び初期値 (Q) を用いて暗号化処理を行う.
Output(V)	V を回答ファイルに出力する.
$U \oplus V$	同じ長さのビット列 U と V との排他的論理和

4.3.3.3 モンテカルロ試験 (MCT) のアルゴリズム

Algorithm 3 MCT Algorithm for MULTI-S01 Encryption

$Key_0 = Key$ ▷ 初期鍵 256 ビット
$PT_0 = PT$ ▷ 初期平文 128 ビット
for $j = 0$ to $outerloop-1$ **do**
  for $i = 0$ to $innerloop-1$ **do**
    $CT_i = $ **MULTI-S01**(Key_j, PT_i, R, IV)
      ▷ 128 ビットの PT_i を暗号化して 256 ビットの暗号文 CT_i を得る.
    $PT_{i+1} = $ **leftmost**$(CT_i, 128)$
  end for
  Output$(CT_{innerloop-1})$ ▷ 回答ファイルに記録する.
  $Key_{j+1} = Key_j \oplus CT_{innerloop-1}$ ▷ 256 ビット分
  $PT_0 = $ **leftmost**$(CT_{innerloop-1}, 128)$ ▷ 128 ビット分
end for

5 確認書発行条件

5.1 パラメータについて

共通鍵暗号において, 暗号アルゴリズム確認書を発行するための条件は, 網掛けされた試験対象機能を少なくとも 1 個実装し, 暗号アルゴリズム実装試験に合格することである. 暗号アルゴリズム実装試験に使用するパラメータの入力条件及びその既定値は, 表 5.1～ 表 5.7 に記載する値とする.

5.1.1 ブロック暗号

5.1.1.1 AES

表 5.1: AES の既定値及び入力条件

アルゴリズム名	試験対象機能	入力欄	既定値	入力条件
AES	暗号化/復号	MMT ブロック数	10	1 以上
		MCT 内側ループ回数	1000	1000 以上
		MCT 外側ループ回数	100	100 以上

5.1.1.2 3-key Triple DES

表 5.2: 3-key Triple DES の既定値及び入力条件

アルゴリズム名	試験対象機能	入力欄	既定値	入力条件
3-key Triple DES	暗号化/復号	MMT ブロック数	10	1 以上
		MCT 内側ループ回数	10000	10000 以上
		MCT 外側ループ回数	400	400 以上

5.1.1.3 その他のブロック暗号

表 5.3: その他のブロック暗号の既定値及び入力条件

ブロック長	試験対象機能	入力欄	既定値	入力条件
128 ビットブロック暗号	暗号化/復号	MMT ブロック数	10	1 以上
		MCT 内側ループ回数	1000	1000 以上
		MCT 外側ループ回数	100	100 以上

5.1.1.4 XTS モード

表 5.4: XTS モードの既定値及び入力条件

アルゴリズム名	試験対象機能	入力欄	既定値	入力条件
XTS モード	暗号化	ブロック暗号	AES	128 ビットブロック暗号
		鍵長	256	256 又は 512
		データユニットサイズ (ビット)	512	8 の倍数かつ 128 以上 2^{16} 以下
		入力データの組の数	100	100 以上
	復号	ブロック暗号	AES	128 ビットブロック暗号
		鍵長	256	256 又は 512
		データユニットサイズ (ビット)	512	8 の倍数かつ 128 以上 2^{16} 以下
		入力データの組の数	100	100 以上

5.1.2 ストリーム暗号

5.1.2.1 KCipher-2

表 5.5: KCipher-2 の既定値及び入力条件

アルゴリズム名	試験対象機能	入力欄		既定値	入力条件
KCipher-2	擬似乱数生成	KAT-Key	ブロック数	24	2 以上 1024 以下
		KAT-IV	ブロック数	24	2 以上 1024 以下
		MCT	内側ループ回数 *innerloop*	1000	1000 以上
			外側ループ回数 *outerloop*	100	100 以上
			N_b	10000	N_c 以上かつ 1000 以上
			N_c	24	4 以上 N_b 以下

5.1.2.2 MUGI

表 5.6: MUGI の既定値及び入力条件

アルゴリズム名	試験対象機能	入力欄		既定値	入力条件
MUGI	擬似乱数生成	KAT-Key	ユニット数	10	250 以下
		MCT	内側ループ回数 *innerloop*	1000	1000 以上
			外側ループ回数 *outerloop*	100	100 以上

5.1.2.3 MULTI-S01

表 5.7: MULTI-S01 暗号の既定値及び入力条件

アルゴリズム名	試験対象機能	入力欄		既定値	入力条件
MULTI-S01	暗号化	KAT-Text	平文のビット長	256	64 の倍数かつ 16000 以下
		MCT	内側ループ回数 *innerloop*	1000	1000 以上
			外側ループ回数 *outerloop*	100	100 以上
	復号	暗号文のビット長		256	64 の倍数かつ 192 以上 16000 以下

附 則
この手順は, 平成 21 年 1 月 23 日から施行し, 平成 21 年 1 月 8 日から適用する.

附 則
この手順は, 平成 21 年 7 月 1 日から施行し, 平成 21 年 7 月 10 日から適用する.

附 則
この手順は, 平成 24 年 2 月 29 日から施行し, 平成 24 年 6 月 1 日から適用する.

附 則
この手順は, 平成 30 年 6 月 22 日から施行し, 平成 30 年 6 月 22 日から適用する.

附 則
この手順は, 令和元年 7 月 11 日から施行し, 令和元年 7 月 11 日から適用する.

参考文献

[1] L. E. Bassham III, *The advanced encryption standard algorithm validation suite (AESAVS)*, National Institute of Standards and Technology, November 15, 2002.

[2] National Institute of Standards and Technology, *Modes of operation validation system for the triple data encryption algorithm (TMOVS) : requirements and procedures*, NIST SP 800-20, April 2000.

[3] National Institute of Standards and Technology, *Recommendation for block cipher modes of operation : methods and techniques*, NIST SP 800-38A, December 2001.

[4] National Institute of Standards and Technology, *Recommendation for Block Cipher Modes of Operation : The XTS-AES Mode for Confidentiality on Storage Devices*, NIST SP 800-38E, January 2010.

[5] IEEE Std 1619-2007, *IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices* 18 April 2008.

[6] Sharon S. Keller and Timothy A. Hall, *The XTS-AES Validation System (XTSVS)*, National Institute of Standards and Technology, March 31, 2010.

[7] KDDI, ストリーム暗号 KCipher-2 (仕様書 1.2 版), http://www.cryptrec.go.jp/cryptrec_13_spec_cypherlist_files/PDF/21_09spec_j_1.2.pdf

[8] JCATT ファイルフォーマット仕様書 – 共通鍵 –, https://www.ipa.go.jp/security/jcmvp/documents/open/jcatt/format/jcatt_fileformat_b.zip

[9] JCATT サンプルファイル – 共通鍵 –, https://www.ipa.go.jp/security/jcmvp/documents/open/jcatt/sample/jcatt_sample_b.zip

改版履歴

識別番号	ATR-01-B

改訂年月日	作成者・承認者	改訂内容
平成 21 年 1 月 23 日	橋本・仲田	新規制定
平成 24 年 2 月 29 日	橋本・仲田	一部改正 (XTS モードに関する試験仕様を追加)
平成 30 年 6 月 22 日	佐伯・江口	一部改正 (承認されたセキュリティ機能の改正に伴い, KCipher-2 の記述を追加し, 128-bit RC4 の記述を削除する. CIPHERUNICORN-E, Hierocrypt-L1, MISTY1, CIPHERUNICORN-A, Hierocrypt-3, SC2000, MUGI, MULTI-S01 を, 暗号アルゴリズム確認対象 非承認セキュリティ機能に移動.)
令和元年 7 月 11 日	櫻井・江口	一部改正 (誤植を訂正)