ACVP Symmetric Block Cipher Algorithm JSON Specification

Christopher Celi
Information Technology Laboratory
Computer Security Division

Russell Hammett
HII Technical Solutions Division

December 10, 2020

National Institute of Standards and Technology
U.S. Department of Commerce

Abstract

This document defines the JSON schema for testing Symmetric Block Cipher implementations with the ACVP specification.

Keywords

The following are keywords to be used by search engines and document catalogues.

ACVP; cryptography

Acknowledgements

The authors thank John Foley for putting together an early draft of this specification.

Disclaimer

Additional Information

Feedback

Feedback on this publication is welcome, and can be sent to: code-signing@nist.gov.

1. Introduction

The Automated Crypto Validation Protocol (ACVP) defines a mechanism to automatically verify the cryptographic implementation of a software or hardware crypto module. The ACVP specification defines how a crypto module communicates with an ACVP server, including crypto capabilities negotiation, session management, authentication, vector processing and more. The ACVP specification does not define algorithm specific JSON constructs for performing the crypto validation. A series of ACVP sub-specifications define the constructs for testing individual crypto algorithms. Each sub-specification addresses a specific class of crypto algorithms. This sub-specification defines the JSON constructs for testing Symmetric Block Cipher implementations using ACVP.

The ACVP server performs a set of tests on the block ciphers in order to assess the correctness and robustness of the implementation. A typical ACVP validation session would require multiple tests to be performed for every supported cryptographic algorithm, such as AES-ECB, AES-CBC, AES-CTR, AES-GCM, TDES-CBC, TDES-CTR, etc.

Unresolved directive in draft-celi-acvp-symmetric.adoc — include::/__w/ACVP-standalone/ACVP-standalone/src/common/common-sections/02-conventions.adoc[]

2. Supported Block Cipher Algorithms

The following block cipher algorithm and test revision pairs **MAY** be advertised by the ACVP compliant cryptographic module (algorithm / revision):

- ACVP-AES-ECB / "1.0"
- ACVP-AES-CBC / "1.0"
- ACVP-AES-CBC-CS1 / "1.0"
- ACVP-AES-CBC-CS2 / "1.0"
- ACVP-AES-CBC-CS3 / "1.0"
- ACVP-AES-OFB / "1.0"
- ACVP-AES-CFB1 / "1.0"
- ACVP-AES-CFB8 / "1.0"
- ACVP-AES-CFB128 / "1.0"
- ACVP-AES-CTR / "1.0"
- ACVP-AES-FF1 / "1.0"
- ACVP-AES-FF3-1 / "1.0"
- ACVP-AES-GCM / "1.0"
- ACVP-AES-GCM-SIV / "1.0"
- ACVP-AES-XPN / "1.0"
- ACVP-AES-CCM / "1.0"
- ACVP-AES-XTS / "1.0"
- ACVP-AES-XTS / "2.0"
- ACVP-AES-KW / "1.0"
- ACVP-AES-KWP / "1.0"
- ACVP-TDES-ECB / "1.0"
- ACVP-TDES-CBC / "1.0"
- ACVP-TDES-CBCI / "1.0"
- ACVP-TDES-CFB1 / "1.0"
- ACVP-TDES-CFB8 / "1.0"
- ACVP-TDES-CFB64 / "1.0"
- ACVP-TDES-CFBP1 / "1.0"
- ACVP-TDES-CFBP8 / "1.0"
- ACVP-TDES-CFBP64 / "1.0"

- ACVP-TDES-OFB / “1.0”
- ACVP-TDES-OFBI / “1.0”
- ACVP-TDES-CTR / “1.0”
- ACVP-TDES-KW / “1.0”

Multiple testing revisions of the same algorithm allow a validation server to improve testing over time without breaking functionality of existing clients. In the case of symmetric block cipher algorithms, multiple revisions of ACVP-AES-XTS exist and **MAY** individually be provided by a server. The revision numbers of “1.0” and “2.0” indicate a progression among the tests. In this case, the “2.0” tests are fully capable of running the “1.0” implementation of ACVP-AES-XTS. More discussion can be found within Section 4.3.

2.1. Supported Conformances

The following conformances **MAY** be advertised by the ACVP compliant cryptographic module:

- ACVP-AES-CCM conformance “ECMA” which expands upon AES-CCM and adheres to [ECMA]. This conformance changes the supported sizes to accomodate the specific sizes required by the use-case of the algorithm.
- ACVP-AES-CTR conformance “RFC3686”. This conformance ensures the IV is generated with the LSB[32] of the IV representing the integer “1”. This conformance also requires the additional registration property of “ivGenMode” which states if the IUT does external or internal IV generation.

3. Test Types and Test Coverage

This section describes the design of the tests used to validate implementations of block cipher algorithms.

3.1. Test Types

There are three types of tests for block ciphers: functional tests, Monte Carlo tests and counter tests. Each has a specific value to be used in the testType field. The testType field definitions are:

- "AFT"—Algorithm Functional Test. These tests can be processed by the client using a normal 'encrypt' or 'decrypt' operation. AFTs cause the implementation under test to exercise normal operations on a single block, multiple blocks, or (where applicable) partial blocks. In some cases random data is used, in others, static, predetermined tests are provided. The functional tests of the block cipher are designed to verify that the logical components of the cipher (GFSbox, KeySbox, block chaining etc.) are operating correctly.
- "MCT"—Monte Carlo Test. These tests exercise the implementation under test under strenuous circumstances. The implementation under test must process the test vectors according to the correct algorithm and mode in this document. MCTs can help detect potential memory leaks over time, and problems in allocation of resources, addressing variables, error handling and generally improper behavior in response to random inputs. Not every algorithm and mode combination has an MCT. See Section 3.1.1 for implementation details.
- "CTR"—Counter Mode Test. Counter tests are specifically for counter modes (AES-CTR and TDES-CTR) and require an implementation under test to exercise their counter mechanism. The server will send a long message to the client for encryption or decryption and back-compute the IVs used by the implementation under test. These IVs are then verified for uniqueness and an increasing (or decreasing) nature. The client processes these tests as normal AFTs. The different mode is highlighted here to signify the difference on the server side for processing.

3.1.1. Monte Carlo tests for block ciphers

The MCTs start with an initial condition (plaintext/ciphertext, key, and optional, or maybe multiple IVs) and perform a series of chained computations. For modes that use an IV, the IV is used in the beginning of each pseudorandom process. The IV is implicitly advanced according to the block cipher mode in use. There are separate rounds of MCT for encryption and decryption. Because some block cipher modes rely on an IV and perform calculations differently from other modes, there are specific definitions of MCT for many of the block cipher modes.

3.1.1.1. AES Monte Carlo Test#—#ECB mode

3.1.1.1.1. Encrypt

The initial condition for the test is the tuple (KEY, PT) set to some values.

The algorithm is shown in Figure 1.

```
Key[0] = KEY
PT[0] = PT
For i = 0 to 99
    Output Key[i]
    Output PT[0]
    For j = 0 to 999
        CT[j] = AES_ECB_ENCRYPT(Key[i], PT[j])
        PT[j+1] = CT[j]
    Output CT[j]
    AES_KEY_SHUFFLE(Key, CT)
    PT[0] = CT[j]
```

Figure 1 — AES-ECB Monte Carlo Test

3.1.1.1.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT’s in the encryption pseudocode with CT’s and all CT’s in the encryption pseudocode with PT’s. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.2. AES Monte Carlo Test#—#CBC mode

3.1.1.2.1. Encrypt

The initial condition for the test is the tuple (KEY, IV, PT) set to some values.

The algorithm is shown in Figure 2.

```
Key[0] = KEY
IV[0] = IV
PT[0] = PT
For i = 0 to 99
    Output Key[i]
    Output IV[i]
    Output PT[0]
    For j = 0 to 999
        If ( j=0 )
            CT[j] = AES_CBC_ENCRYPT(Key[i], IV[i], PT[j])
            PT[j+1] = IV[i]
        Else
            CT[j] = AES_CBC_ENCRYPT(Key[i], PT[j])
            PT[j+1] = CT[j-1]
    Output CT[j]
    AES_KEY_SHUFFLE(Key, CT)
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
```

Figure 2 — AES-CBC Monte Carlo Test

3.1.1.2.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.3. AES Monte Carlo Test#—#OFB mode

3.1.1.3.1. Encrypt

The initial condition for the test is the tuple (KEY, IV, PT) set to some values.

The algorithm is shown in Figure 3.

```
Key[0] = Key
IV[0] = IV
PT[0] = PT
For i = 0 to 99
    Output Key[i]
    Output IV[i]
    Output PT[0]
    For j = 0 to 999
        If ( j=0 )
            CT[j] = AES_OFB_ENCRYPT(Key[i], IV[i], PT[j])
            PT[j+1] = IV[i]
        Else
            CT[j] = AES_OFB_ENCRYPT(Key[i], PT[j])
            PT[j+1] = CT[j-1]
    Output CT[j]
    AES_KEY_SHUFFLE(Key, CT)
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
```

Figure 3 — AES-OFB Monte Carlo Test

3.1.1.3.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.4. AES Monte Carlo Test#—#CFB1 mode

3.1.1.4.1. Encrypt

The initial condition for the test is the tuple (KEY, IV, PT) set to some values.

The algorithm is shown in Figure 4.

```
Key[0] = Key
IV[0] = IV
PT[0] = PT
For i = 0 to 99
    Output Key[i]
    Output IV[i]
    Output PT[0]
    For j = 0 to 999
        If ( j=0 )
            CT[j] = AES_CFB1_ENCRYPT(Key[i], IV[i], PT[j])
            PT[j+1] = BitJ(IV[i])
        Else
            CT[j] = AES_CFB1_ENCRYPT(Key[i], PT[j])
            If ( j<128 )
                PT[j+1] = BitJ(IV[i])
            Else
                PT[j+1] = CT[j-128]
    Output CT[j]
    If ( keylen = 128 )
        Key[i+1] = Key[i] xor (CT[j-127] || CT[j-126] || ... || CT[j])
    If ( keylen = 192 )
        Key[i+1] = Key[i] xor (CT[j-191] || CT[j-190] || ... || CT[j])
    If ( keylen = 256 )
        Key[i+1] = Key[i] xor (CT[j-255] || CT[j-254] || ... || CT[j])
    IV[i+1] = (CT[j-127] || CT[j-126] || ... || CT[j])
    PT[0] = CT[j-128]
```

Figure 4 — AES-CFB1 Monte Carlo Test

3.1.1.4.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.5. AES Monte Carlo Test — CFB8 mode

3.1.1.5.1. Encrypt

The initial condition for the test is the tuple (KEY, IV, PT) set to some values.

The algorithm is shown in Figure 5.

```
Key[0] = Key
IV[0] = IV
PT[0] = PT
For i = 0 to 99
    Output Key[i]
    Output IV[i]
    Output PT[0]
    For j = 0 to 999
        If ( j=0 )
            CT[j] = AES_CFB8_ENCRYPT(Key[i], IV[i], PT[j])
            PT[j+1] = ByteJ(IV[i])
        Else
            CT[j] = AES_CFB8_ENCRYPT(Key[i], PT[j])
            If ( j<16 )
                PT[j+1] = ByteJ(IV[i])
            Else
                PT[j+1] = CT[j-16]
    Output CT[j]
    If ( keylen = 128 )
        Key[i+1] = Key[i] xor (CT[j-15] || CT[j-14] || ... || CT[j])
    If ( keylen = 192 )
        Key[i+1] = Key[i] xor (CT[j-23] || CT[j-22] || ... || CT[j])
    If ( keylen = 256 )
        Key[i+1] = Key[i] xor (CT[j-31] || CT[j-30] || ... || CT[j])
    IV[i+1] = (CT[j-15] || CT[j-14] || ... || CT[j])
    PT[0] = CT[j-16]
```

Figure 5 — AES-CFB8 Monte Carlo Test

3.1.1.5.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.6. AES Monte Carlo Test – CFB128 mode

3.1.1.6.1. Encrypt

The initial condition for the test is the tuple (KEY, IV, PT) set to some values.

The algorithm is shown in Figure 6.

```
Key[0] = Key
IV[0] = IV
PT[0] = PT
For i = 0 to 99
    Output Key[i]
    Output IV[i]
    Output PT[0]
    For j = 0 to 999
        If ( j=0 )
            CT[j] = AES_CFB128_ENCRYPT(Key[i], IV[i], PT[j])
            PT[j+1] = IV[i]
        Else
            CT[j] = AES_CFB128_ENCRYPT(Key[i], PT[j])
            PT[j+1] = CT[j-1]
    Output CT[j]
    AES_KEY_SHUFFLE(Key, CT)
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
```

Figure 6 — AES-CFB128 Monte Carlo Test

3.1.1.6.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.7. AES Monte Carlo Key Shuffle

Most AES MCTs use a shared key shuffle routine. The algorithm is shown in Figure 7.

The initial condition for the routine is a tuple (KEY, CT) set to some values. This pseudocode is specifically for encryption. For decryption, swap all instances of CT with PT. The || symbol is used to denote concatenation. The MSB (most significant bits) and LSB (least significant bits) functions accept a bit string and an integer amount of bits to capture. For example MSB(A, 8) would capture the 8 most significant bits of the bit string A.

```
If ( keylen = 128 )
```

```
    Key[i+1] = Key[i] xor MSB(CT[j], 128)
If ( keylen = 192 )
    Key[i+1] = Key[i] xor (LSB(CT[j-1], 64) || MSB(CT[j], 128))
If ( keylen = 256 )
    Key[i+1] = Key[i] xor (MSB(CT[j-1], 128) || MSB(CT[j], 128))
```

Figure 7 — AES Encrypt Key Shuffle Routine

3.1.1.8. TDES Monte Carlo Test#—#ECB mode

3.1.1.8.1. Encrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, PT) set to some values.

The algorithm is shown in Figure 8.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
PT[0] = PT
For i = 0 to 399
    Output Key1[i]
    Output Key2[i]
    Output Key3[i]
    Output PT[0]
    For j = 0 to 9999
        CT[j] = TDES_ECB_ENCRYPT(Key1[i], Key2[i], Key3[i], PT[j])
        PT[j+1] = CT[i]
    Output CT[j]
    Key1[i+1] = Key1[i] xor CT[j]
    Key2[i+1] = Key2[i] xor CT[j-1]
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor CT[j-2]
    Else
        Key3[i+1] = Key1[i+1]
    PT[0] = CT[j]
```

Figure 8 — TDES-ECB Monte Carlo Test

3.1.1.8.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT’s in the encryption pseudocode with CT’s and all CT’s in the encryption pseudocode with PT’s. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.9. TDES Monte Carlo Test#—#CBC mode

3.1.1.9.1. Encrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV, PT) set to some values.

The algorithm is shown in Figure 9.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV[0] = IV
PT[0] = PT
For i = 0 to 399
    Output Key1[i]
    Output Key2[i]
    Output Key3[i]
    Output IV[0]
    Output PT[0]
    For j = 0 to 9999
        CT[j] = TDES_CBC_ENCRYPT(Key1[i], Key2[i], Key3[i], PT[j], IV[j])
        If ( j = 0 )
            PT[j+1] = IV[0]
        Else
            PT[j+1] = CT[j-1]
        IV[j+1] = CT[j]
    Output CT[j]
    Key1[i+1] = Key1[i] xor CT[j]
    Key2[i+1] = Key2[i] xor CT[j-1]
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor CT[j-2]
    Else
        Key3[i+1] = Key1[i+1]
    PT[0] = CT[j-1]
    IV[0] = CT[j]
```

Figure 9 — TDES-CBC Monte Carlo Test

3.1.1.9.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the inner loop in the pseudocode with the following:

```
    For j = 0 to 9999
```

```
        PT[j] = TDES_CBC_DECRYPT(Key1[i], Key2[i], Key3[i], CT[j], IV[j])
        CT[j+1] = PT[j]
        IV[j+1] = CT[j]
```

Figure 10 — TDES-CBC Monte Carlo Test Decrypt

3.1.1.10. TDES Monte Carlo Test – CBC-I mode

3.1.1.10.1. Encrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV1, IV2, IV3, PT1, PT2, PT3) set to some values.

The algorithm is shown in Figure 11.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV1[0] = IV1
IV2[0] = IV2
IV3[0] = IV3
PT1[0] = PT1
PT2[0] = PT2
PT3[0] = PT3
For i = 0 to 399
    Output Key1[i], Key2[i], Key3[i]
    Output IV1[0], IV2[0], IV3[0]
    Output PT1[0], PT2[0], PT3[0]
    For j = 0 to 9999
        CT[j] = TDES_CBC_I_ENCRYPT(Key1[i], Key2[i], Key3[i], PT1[j], PT2[j],
 PT3[j], IV1[j], IV2[j], IV3[j])
        If ( j = 0 )
            PT1[j+1] = IV1[0]
            PT2[j+1] = IV2[0]
            PT3[j+1] = IV3[0]
        Else
            PT1[j+1] = CT1[j-1]
            PT2[j+1] = CT2[j-1]
            PT3[j+1] = CT3[j-1]
        IV1[j+1] = CT1[j]
        IV2[j+1] = CT2[j]
        IV3[j+1] = CT3[j]
    Output CT1[j], CT2[j], CT3[j]
    Key1[i+1] = Key1[i] xor CT1[j]
    Key2[i+1] = Key2[i] xor CT2[j-1]
    If ( keyingOption = 1 )
```

```
        Key3[i+1] = Key3[i] xor CT3[j-2]
    Else
        Key3[i+1] = Key1[i+1]
    PT1[0] = CT1[j-1]
    PT2[0] = CT2[j-1]
    PT3[0] = CT3[j-1]
    IV1[0] = CT1[j]
    IV2[0] = CT2[j]
    IV3[0] = CT3[j]
```

Figure 11 — TDES-CBC-I Monte Carlo Test

3.1.1.10.2. Decrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV1, IV2, IV3, CT1, CT2, CT3) set to some values.

The algorithm is shown in Figure 12.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV1[0] = IV1
IV2[0] = IV2
IV3[0] = IV3
CT1[0] = CT1
CT2[0] = CT2
CT3[0] = CT3
For i = 0 to 399
    Output Key1[i], Key2[i], Key3[i]
    Output IV1[0], IV2[0], IV3[0]
    Output CT1[0], CT2[0], CT3[0]
    For j = 0 to 9999
        PT[j] = TDES_CBC_I_DECRYPT(Key1[i], Key2[i], Key3[i], CT1[j], CT2[j],
 CT3[j], IV1[j], IV2[j], IV3[j])
        CT1[j+1] = PT1[j]
        CT2[j+1] = PT2[j]
        CT3[j+1] = PT3[j]
        IV1[j+1] = CT1[j]
        IV2[j+1] = CT2[j]
        IV3[j+1] = CT3[j]
    Output PT1[j], PT2[j], PT3[j]
    Key1[i+1] = Key1[i] xor PT1[j]
    Key2[i+1] = Key2[i] xor PT2[j-1]
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor PT3[j-2]
    Else
        Key3[i+1] = Key1[i+1]
    CT1[0] = PT1[j]
    CT2[0] = PT2[j]
    CT3[0] = PT3[j]
    IV1[0] = CT1[j]
    IV2[0] = CT2[j]
    IV3[0] = CT3[j]
```

Figure 12 — TDES-CBC-I Monte Carlo Test Decrypt

3.1.1.11. TDES Monte Carlo Test#—#CFB1, CFB8, CFB64 modes

3.1.1.11.1. Encrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV, PT) set to some values. PT and CT are k-bit where k is the feedback size, for example CFB1 has a feedback size of 1-bit.

The algorithm is shown in Figure 13.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV[0] = IV
PT[0] = PT
For i = 0 to 399
    Output Key1[i]
    Output Key2[i]
    Output Key3[i]
    Output IV[0]
    Output PT[0]
    For j = 0 to 9999
        CT[j] = TDES_CFB_ENCRYPT(Key1[i], Key2[i], Key3[i], PT[j], IV[j])
        PT[j+1] = LeftMost_K_Bits(IV[j])
        IV[j+1] = RightMost_64-K_Bits(IV[j]) || CT[j]
    Output CT[j]
    C = LeftMost_192_Bits(CT[j] || CT[j-1] || ... || CT[0])
    Key1[i+1] = Key1[i] xor bits 129-192 of C
    Key2[i+1] = Key2[i] xor bits 65-128 of C
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor bits 1-64 of C
    Else
        Key3[i+1] = Key1[i+1]
    PT[0] = LeftMost_K_Bits(IV[j])
    IV[0] = RightMost_64-K_Bits(IV[j]) || CT[j]
```

Figure 13 — TDES-CFB Monte Carlo Test

3.1.1.11.2. Decrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV, CT) set to some values. PT and CT are k-bit where k is the feedback size, for example CFB1 has a feedback size of 1-bit. O[j] is the O_j variable internal to the Triple DES operation described in Table 43 of SP 800-20.

The algorithm is shown in Figure 14.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV[0] = IV
CT[0] = CT
For i = 0 to 399
    Output Key1[i]
    Output Key2[i]
    Output Key3[i]
    Output IV[0]
    Output CT[0]
    For j = 0 to 9999
        PT[j] = TDES_CFB_DECRYPT(Key1[i], Key2[i], Key3[i], CT[j], IV[j])
        CT[j+1] = LeftMost_K_Bits(O[j])
        IV[j+1] = RightMost_64-K_Bits(IV[j]) || CT[j]
    Output PT[j]
    C = LeftMost_192_Bits(PT[j] || PT[j-1] || ... || PT[0])
    Key1[i+1] = Key1[i] xor bits 129-192 of C
    Key2[i+1] = Key2[i] xor bits 65-128 of C
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor bits 1-64 of C
    Else
        Key3[i+1] = Key1[i+1]
    CT[0] = LeftMost_K_Bits(O[j])
    IV[0] = RightMost_64-K_Bits(IV[j]) || CT[j]
```

Figure 14 — TDES-CFB Monte Carlo Test Decrypt

3.1.1.12. TDES Monte Carlo Test#—#CFB1-P, CFB8-P, CFB64-P modes

3.1.1.12.1. Encrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV1, IV2, IV3, PT) set to some values. PT and CT are k-bit where k is the feedback size, for example CFB8-P has a feedback size of 8-bits.

The algorithm is shown in Figure 15.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV1[0] = IV1
IV2[0] = IV2
IV3[0] = IV3
PT[0] = PT
For i = 0 to 399
    Output Key1[i], Key2[i], Key3[i]
    Output IV1[0]
    Output PT[0]
    For j = 0 to 9999
        CT[j] = TDES_CFB_P_ENCRYPT(Key1[i], Key2[i], Key3[i], PT[j], IV1[j],
 IV2[j], IV3[j])
        PT[j+1] = LeftMost_K_Bits(IV1[j]) <-- This line may not be correct?
 Compare to SP 800-20 Table 49
    Output CT[j]
    C = LeftMost_192_Bits(CT[j] || CT[j-1] || ... || CT[0])
    Key1[i+1] = Key1[i] xor bits 129-192 of C
    Key2[i+1] = Key2[i] xor bits 65-128 of C
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor bits 1-64 of C
    Else
        Key3[i+1] = Key1[i+1]
    PT[0] = LeftMost_K_Bits(IV1[j])
    IV1[0] = RightMost_64-K_Bits(IV[j]) || CT[j]
    IV2[0] = IV1[0] + "5555555555555555" mod 2^64
    IV3[0] = IV1[0] + "AAAAAAAAAAAAAAAA" mod 2^64
```

Figure 15 — TDES-CFB-P Monte Carlo Test

3.1.1.12.2. Decrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV1, IV2, IV3, CT) set to some values. PT and CT are k-bit where k is the feedback size, for example CFB8-P has a feedback size of 8-bits. O[j] is the O_j variable internal to the Triple DES operation described in Table 50 of SP 800-20.

The algorithm is shown in Figure 16.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV1[0] = IV1
IV2[0] = IV2
IV3[0] = IV3
CT[0] = CT
For i = 0 to 399
    Output Key1[i], Key2[i], Key3[i]
    Output IV1[0]
    Output CT[0]
    For j = 0 to 9999
        PT[j] = TDES_CFB_P_DECRYPT(Key1[i], Key2[i], Key3[i], CT[j], IV1[j],
 IV2[j], IV3[j])
        CT[j+1] = LeftMost_K_Bits(O[j])
    Output PT[j]
    C = LeftMost_192_Bits(PT[j] || PT[j-1] || ... || PT[0])
    Key1[i+1] = Key1[i] xor bits 129-192 of C
    Key2[i+1] = Key2[i] xor bits 65-128 of C
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor bits 1-64 of C
    Else
        Key3[i+1] = Key1[i+1]
    CT[0] = LeftMost_K_Bits(O[j])
    IV1[0] = RightMost_64-K_Bits(IV[j]) || CT[j]
    IV2[0] = IV1[0] + "5555555555555555" mod 2^64
    IV3[0] = IV1[0] + "AAAAAAAAAAAAAAAA" mod 2^64
```

Figure 16 — TDES-CFB-P Monte Carlo Test Decrypt

3.1.1.13. TDES Monte Carlo Test#—#OFB mode

3.1.1.13.1. Encrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV, PT) set to some values.

The algorithm is shown in Figure 17.

```
Key1[0] = KEY1
```

```
Key2[0] = KEY2
Key3[0] = KEY3
IV[0] = IV
PT[0] = PT
For i = 0 to 399
    Output Key1[i]
    Output Key2[i]
    Output Key3[i]
    Output IV[0]
    Output PT[0]
    For j = 0 to 9999
        CT[j] = TDES_OFB_ENCRYPT(Key1[i], Key2[i], Key3[i], PT[j], IV[j])
        PT[j+1] = IV[j]
    Output CT[j]
    Key1[i+1] = Key1[i] xor CT[j]
    Key2[i+1] = Key2[i] xor CT[j-1]
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor CT[j-2]
    Else
        Key3[i+1] = Key1[i+1]
    PT[0] = PT[0] xor IV[j]
    IV[0] = CT[j]
```

Figure 17 — TDES-OFB Monte Carlo Test

3.1.1.13.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the encrypt operation with the corresponding decrypt operation.

3.1.1.14. TDES Monte Carlo Test#—#OFB-I mode

3.1.1.14.1. Encrypt

The initial condition for the test is the tuple (KEY1, KEY2, KEY3, IV1, IV2, IV3, PT) set to some values.

The algorithm is shown in Figure 18.

```
Key1[0] = KEY1
Key2[0] = KEY2
Key3[0] = KEY3
IV1[0] = IV1
IV2[0] = IV2
IV3[0] = IV3
PT[0] = PT
```

```
For i = 0 to 399
    Output Key1[i], Key2[i], Key3[i]
    Output IV1[0], IV2[0], IV3[0]
    Output PT[0]
    For j = 0 to 9999
        CT[j] = TDES_OFB-I_ENCRYPT(Key1[i], Key2[i], Key3[i], PT[j], IV[j])
        PT[j+1] = IV[j]
    Output CT[j]
    Key1[i+1] = Key1[i] xor CT[j]
    Key2[i+1] = Key2[i] xor CT[j-1]
    If ( keyingOption = 1 )
        Key3[i+1] = Key3[i] xor CT[j-2]
    Else
        Key3[i+1] = Key1[i+1]
    PT[0] = PT[0] xor IV1[j]
    IV1[0] = CT[j]
    IV2[0] = IV1[0] + "5555555555555555" mod 2^64
    IV3[0] = IV1[0] + "AAAAAAAAAAAAAAAA" mod 2^64
```

Figure 18 — TDES-OFB-I Monte Carlo Test

3.1.1.14.2. Decrypt

The pseudocode for decryption can be obtained by replacing all PT's in the encryption pseudocode with CT's and all CT's in the encryption pseudocode with PT's. As well, replace the encrypt operation with the corresponding decrypt operation.

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [FIPS 197] and [SP 800-38A].

3.2.1. AES Requirements Covered

In [SP 800-38A], both Section 5 and Section 6 which describe general modes of operation for block ciphers are tested. In [FIPS 197], Section 4 outlines the AES engine and necessary functions to perform simple encrypt an decrypt operations. All AES tests perform such operations and thus rely heavily on this section. Section 5 specifically outlines the algorithm for AES and thus all AES tests rely heavily on this section as well. All of [SP 800-38A] requirements are covered. In [IEEE 1619-2007], the IEEE outlines the encrypt and decrypt operations for AES-XTS.

3.2.2. AES Requirements Not Covered

Some requirements in the outlined specifications are not easily tested. Often they are not ideal for black-box testing such as the ACVP. In [SP 800-38A], Appendix A outlines padding for when the data being encrypted does not evenly fill the blocks. In these tests, all data, unless otherwise specified, is assumed to be a multiple of the block length. All exceptions to those cases are when stream ciphers specifically are being tested. In Section 5.3, IV generation which is required for all modes of AES and TDES outside of ECB, is not tested. Appendix D outlines how errors are to be handled. As some symmetric ciphers aren't authenticated, ACVP does not include tests that change random bits in payload, IV, key or results, as these results can be successfully encrypted/decrypted, but errors aren't necessarily detectable.

In [FIPS 197], Section 5.3 defines the inverse cipher for AES. This is not tested in the CBC, CFB (all), OFB or CTR modes.

In [SP 800-38E], the AES-XTS algorithm is restricted to 2\^20 AES blocks (128-bits each) per key. Due to the size of the data, ACVP does not test the proper usage of a key over such large amounts of data.

In the [RFC 3686] testing conformance of AES-CTR, tests will be generated ensuring the LSB[32] of the IV represents the integer value of "1". These tests will allow for either internal or external IV generation from the perspective of the IUT.

3.2.3. AES Format Preserving Encryption Requirements Covered

All of [SP 800-38G] requirements are covered.

3.2.4. AES Format Preserving Encryption Requirements Not Covered

N/A

3.2.5. TDES Requirements Covered

In [SP 800-67 Rev. 2], Section 3 outlines the use for TDES with keying option 1 (three distinct keys) and decryption only for keying option 2 (K1 == K3 != K2). Depending on the cipher mode, both the forward and inverse cipher are tested. The known answer tests address these requirements.

3.2.6. TDES Requirements Not Covered

In [SP 800-67 Rev. 2], Section 3.3 outlines requirements for keys for proper usage of TDES. These requirements are not tested by ACVP. All keys used in the tests are randomly or staticly generated by the server. There are no checks for key equality or potentially weak keys. Section 3.3.2 outlines specific keys which are to be avoided. ACVP does not expect a client to be able to detect these keys.

3.2.7. AEAD Requirements Covered

In [SP 800-38D], Section 7 outlines the encrypt and decrypt operations for AES-GCM. This and all prerequisites to these operations (such as GHASH) are tested as AES-GCM encrypt and decrypt operations.

In [SP 800-38C], Section 6 outlines the encrypt and decrypt operations for AES-CCM. This and all prerequisites to these operations (such as CBC-MAC) are tested as AES-CCM encrypt and decrypt operations. In [AES-GCM-SIV], the draft outlines the encrypt and decrypt operations for AES-GCM-SIV.

3.2.8. AEAD Requirements Not Covered

In [SP 800-38D], Section 8 outlines uniqueness requirements on IVs and keys for AES-GCM. This is considered out of bounds for the algorithm testing done by the ACVP and will not be tested.

3.2.9. KeyWrap Requirements Covered

In [SP 800-38F] Section 5.2 defines the authenticated encryption and authenticated decryption operations for all three key-wrap algorithms. As well, the padding for key-wrap with padding is defined. Algorithm Functional Tests provide assurance of these requirements for encrypt operations. For decrypt operations, there is a possibility to reject the ciphertext due to improper wrapping. This is also assured by the Algorithm Functional Tests.

Sections 6 and 7 outline the specific ciphers in both encrypt and decrypt directions. All facsets of these processes are tested with random data via the Algorithm Functional Tests.

3.2.10. KeyWrap Requirements Not Covered

In [SP 800-38F] Section 5.3 defines the length requirements allowed by an optimal implementation. The upper bounds are unreasonably large to test in a web-based model and thus an artificial maximum is selected for the payloadLen property (corresponding to both plaintext and ciphertext). The Algorithm Functional Tests SHOULD utilize both the minimum and maximum values provided in the client’s registration optimally with other values.

4. Capabilities Registration

ACVP requires crypto modules to register their capabilities. This allows the crypto module to advertise support for specific algorithms, notifying the ACVP server which algorithms need test vectors generated for the validation process. This section describes the constructs for advertising support of Block Cipher algorithms to the ACVP server.

The algorithm capabilities **MUST** be advertised as JSON objects within the 'algorithms' value of the ACVP registration message. The 'algorithms' value is an array, where each array element is an individual JSON object defined in this section. The 'algorithms' value is part of the 'capability_exchange' element of the ACVP JSON registration message. See the ACVP specification [ACVP] for more details on the registration message.

4.1. Prerequisites

Each algorithm implementation **MAY** rely on other cryptographic primitives. For example, RSA Signature algorithms depend on an underlying hash function. Each of these underlying algorithm primitives must be validated, either separately or as part of the same submission. ACVP provides a mechanism for specifying the required prerequisites:

Prerequisites, if applicable, **MUST** be submitted in the registration as the `prereqVals` JSON property array inside each element of the `algorithms` array. Each element in the `prereqVals` array **MUST** contain the following properties

Table 1 — Prerequisite Properties

JSON Property	Description	JSON Type
algorithm	a prerequisite algorithm	string
valValue	algorithm validation number	string

A "valValue" of "same" **SHALL** be used to indicate that the prerequisite is being met by a different algorithm in the capability exchange in the same registration.

An example description of prerequisites within a single algorithm capability exchange looks like this

```
"prereqVals":
[
  {
    "algorithm": "Alg1",
    "valValue": "Val-1234"
  },
  {
    "algorithm": "Alg2",
    "valValue": "same"
  }
]
```

Figure 19

4.2. Required Prerequisite Algorithms for Block Cipher Validations

Some block cipher algorithm implementations rely on other cryptographic primitives. For example, AES-CCM uses an underlying AES-ECB algorithm. Each of these underlying algorithm primitives **MUST** be validated, either separately or as part of the same submission. ACVP provides a mechanism for specifying the required prerequisites:

Table 2 — Required Prerequisite Algorithms JSON Values

JSON Value	Description	JSON type	Example Values
algorithm	a prerequisite algorithm	string	AES, DRBG, TDES
valValue	algorithm validation number	string	actual number or “same” to refer to the same submission
prereqAlgVal	prerequisite algorithm validation	object	exactly one algorithm property and one valValue property

4.3. Block Cipher Algorithm Capabilities JSON Values

Each algorithm capability advertised is a self-contained JSON object and **SHALL** use the following values when appropriate:

Table 3 — Block Cipher Algorithm Capabilities JSON Values

JSON Value	Description	JSON type
algorithm	The block cipher algorithm and mode to be validated.	string
revision	The version of the testing methodology the IUT is requesting to validate against.	string
conformances	The additional conformance for the algorithm for a specific use-case	array of string
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects described in Table 2
direction	The IUT processing direction	array of strings
keyLen	The supported key lengths in bits	array of integers

JSON Value	Description	JSON type
payloadLen	The supported plain and cipher text lengths in bits. This varies depending on the algorithm type; for additional details see Table 5, Table 7, Table 8 and Table 9. For AES-CTR, the values supplied for this parameter refer to the bit sizes supported in the last incomplete block (less than 128 bits) of the plain or cipher text.	domain
ivLen	The supported IV/Nonce lengths in bits, see Table 7	domain
ivGen	IV generation method for AES-GCM/AES-XPN algorithms	string
ivGenMode	IV generation mode for AES-GCM/AES-XPN algorithms	string
saltGen	Salt generation method for AES-XPN mode only	string
aadLen	The supported AAD lengths in bits for AEAD algorithms	domain
tagLen	The supported Tag lengths in bits for AEAD algorithms, see Table 7	array of integers
kwCipher	The cipher as defined in SP800-38F for key wrap mode	array of strings
tweakMode	Indicates the format(s) of the tweak value input for AES-XTS. A value of “hex” indicates that the IUT expects the tweak value input as a hexadecimal string. A value of “number” indicates that the IUT expects to receive a Data Unit Sequence Number.	array of strings

JSON Value	Description	JSON type
keyingOption	The Keying Option used in TDES. Keying option 1 (1) is 3 distinct keys (K1, K2, K3). Keying Option 2 (2) is 2 distinct keys only suitable for decrypt (K1, K2, K1).	array of integers
overflowCounter	Indicates if the implementation can handle a counter exceeding the maximum value	boolean
incrementalCounter	Indicates if the implementation increments the counter (versus decrementing the counter)	boolean
performCounterTests	Indicates if the implementation can perform the Counter tests which check for an always increasing (or decreasing) counter value	boolean
tweakLen	The domain of values allowed for ACVP-AES-FF1's tweak value. Allowed range is 0-128 bits mod 8. See Table 9	domain
capabilities	An array of objects that describes an IUT's capabilities as they pertain to ACVP-AES-FF1 and ACVP-AES-FF3-1. See Table 9 and Table 10	Array of Objects
dataUnitLen	The length of the ACVP-AES-XTS data unit	domain
dataUnitLenMatchesPayload	Whether or not the length of the data unit always matches the length of the	boolean

JSON Value	Description	JSON type
	payload in ACVP-AES-XTS	

NOTE 1 – The 'conformances' property is only valid for algorithms listed in Section 2.1. The valid values in the array are also listed in that section. The array is always optional.

NOTE 2 – Some optional values are required depending on the algorithm. For example, AES-GCM requires ivLen, ivGen, ivGenMode, aadLen and tagLen. Failure to provide these values will result in the ACVP server returning an error to the ACVP client during registration.

NOTE 3 – The 'performCounterTests' option is provided for counter implementations such as linear-feedback shift registers which may not present an always increasing or decreasing counter while still ensuring the IV is unique. This value defaults to true if not present. If it is set to false, the 'overflowCounter' and 'incrementalCounter' values will not be used.

The following grid outlines which properties are **REQUIRED**, as well as all the possible values a server **MAY** support for each standard block cipher algorithm:

Table 4 — Standard Block Cipher Algorithm Capabilities Applicability Grid

algorithm	revision	direction	keyLen	keyingOption
AES-ECB	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	
AES-CBC	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	
AES-OFB	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	
AES-CFB1	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	
AES-CFB8	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	
AES-CFB128	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	
TDES-ECB	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-CBC	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-CBCI	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-CFB1	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations

algorithm	revision	direction	keyLen	keyingOption
TDES-CFB8	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-CFB64	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-CFBP1	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-CFBP8	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-CFBP64	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-OFB	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
TDES-OFBI	"1.0"	["encrypt", "decrypt"]		[1, 2] Note: 2 is only available for decrypt operations
NOTE – keyingOption 2 **SHALL** only be available for decrypt operations.				

The following grid outlines which properties are **REQUIRED**, as well as the possible values a server **MAY** support for each key-wrap block cipher algorithm:

Table 5 — Key-Wrap Block Cipher Algorithm Capabilities Applicability Grid

algorithm	revision	direction	keyLen	kwCipher	keyingOption	payloadLen
AES-KW	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	["cipher", "inverse"]		{"Min": 128, "Max": 4096, "Increment": 64}
AES-KWP	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	["cipher", "inverse"]		{"Min": 8, "Max": 4096, "Increment": 8}
TDES-KW	"1.0"	["encrypt", "decrypt"]		["cipher", "inverse"]	[1, 2] Note: 2 is only available for decrypt operations	{"Min": 64, "Max": 4096, "Increment": 32}

The underlying operations associated with different KW and KWP parameter selections are summarized in the following grid.

Table 6 — Wrapping and Unwrapping Operations

Operation	Cipher	Underlying AES Operation
Wrap (direction encrypt)	Cipher	AES Encrypt
Wrap (direction encrypt)	Inverse	AES Decrypt
Unwrap (direction decrypt)	Cipher	AES Decrypt
Unwrap (direction decrypt)	Inverse	AES Encrypt

The following grid outlines which properties are **REQUIRED**, as well as the possible values a server **MAY** support for each authenticated block cipher algorithm:

Table 7 — Authenticated Block Cipher Algorithm Capabilities Applicability Grid

algorithm	revision	direction	keyLen	payloadLen	ivLen	ivGen	ivGenMode	saltGen	aadLen	tagLen
AES-GCM	“1.0”	[“encrypt”, “decrypt”]	[128, 192, 256]	{”Min”: 0, “Max”: 65536, “Inc”: any}	{”Min”: 8, “Max”: 1024, “Inc”: any}	[“internal”, “external”]	[“8.2.1”, “8.2.2”]		{”Min”: 0, “Max”: 65536, “Inc”: any}	[32, 64, 96, 104, 112, 120, 128]
AES-GCM-SIV	“1.0”	[“encrypt”, “decrypt”]	[128, 256]	{”Min”: 0, “Max”: 65536, “Inc”: 8}					{”Min”: 0, “Max”: 65536, “Inc”: 8}	
AES-XPN	“1.0”	[“encrypt”, “decrypt”]	[128, 192, 256]	{”Min”: 0, “Max”: 65536, “Inc”: any}		[“internal”, “external”]	[“8.2.1”, “8.2.2”]	[“internal”, “external”]	{”Min”: 1, “Max”: 65536, “Inc”: any}	[32, 64, 96, 104, 112, 120, 128]
AES-CCM	“1.0”	[“encrypt”, “decrypt”]	[128, 192, 256]	{”Min”: 0, “Max”: 256, “Inc”: 8}	{”Min”: 56, “Max”: 104, “Inc”: 8}				{”Min”: 0, “Max”: 524288, “Inc”: any}	[32, 48, 64, 80, 96, 112, 128]
NOTE – The ivGenMode property is used for AES-GCM/AES-XPN algorithms to document which IV construction method the implementation conforms to: the deterministic construction defined in SP 800-38D section 8.2.1 or the RBG-based construction defined in SP 800-38D section 8.2.2.										

The following grid outlines which properties are **REQUIRED**, as well as the possible values a server **MAY** support for the XTS block cipher algorithm:

Table 8 — XTS Block Cipher Algorithm Capabilities Applicability Grid

algorithm	revision	direction	keyLen	payloadLen	tweakMode	dataUnitLen	dataUnitLenMatchesPayload
ACVP-AES-XTS	"1.0"	["encrypt", "decrypt"]	[128, 256]	{"Min": 128, "Max": 65536, "Inc": 128}	["hex", "number"]		
ACVP-AES-XTS	"2.0"	["encrypt", "decrypt"]	[128, 256]	{"Min": 128, "Max": 65536, "Inc": 8}	["hex", "number"]	{"Min": 128, "Max": 65536, "Inc": 8}	true, false (if this value is true, the dataUnitLen parameter **SHALL** not be present; if this value is false, the dataUnitLen parameter **SHALL** be present)
NOTE – The difference in testing between ACVP-AES-XTS / "1.0" and ACVP-AES-XTS / "2.0" is the inclusion of the data unit in the "2.0" revision. The [IEEE 1619-2007] standard provides the concept of a data unit as a means of logically breaking apart a data stream provided to the encryption algorithm. A data unit may be larger, smaller or equal to the payload being processed. In the case of the "1.0" revision, the data unit length always matches the payload length. Thus, the "1.0" revision can be accessed via the "2.0" revision by setting the 'dataUnitLenMatchesPayload' field to true. Within the prompt, in "1.0", the test group contains the payload length for the entire group. In "2.0" this is moved to the test case level and handled on a per case basis along with the data unit length. Both values may be provided even when 'dataUnitLenMatchesPayload' is true.							

The following grid outlines which properties are **REQUIRED**, as well as the possible values a server **MAY** support for each miscellaneous block cipher algorithm:

Table 9 — Miscellaneous Block Cipher Algorithm Capabilities Applicability Grid

lgorithn	revisior	directior	keyLe	payloadLe	keyingOptio	overflowCoun	ncrementalCoun	erformCounterTe	weakLe	capabilities
AES-CBC-CS1	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	{"Min": 128, "Max": 65536, "Inc": any}						
AES-CBC-CS2	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	{"Min": 128, "Max": 65536, "Inc": any}						
AES-CBC-CS3	"1.0"	["encrypt", "decrypt"]	[128, 192, 256]	{"Min": 128, "Max": 65536,						

algorith	revisio	direction	keyLe	payloadL	keyingOptio	overflowCount	ncrementalCou	erformCounterTe	weakLe	capabilities
				“Inc”: any}						
AES-CTR	“1.0”	[“encrypt”, “decrypt”]	[128, 192, 256]	{”Min”: 1, “Max”: 128, “Inc”: any}		true, false	true, false	true, false		
TDES-CTR	“1.0”	[“encrypt”, “decrypt”]		{”Min”: 1, “Max”: 64, “Inc”: any}	[1, 2] Note: 2 is only available for decrypt operations	true, false	true, false	true, false		
AES-FF1	“1.0”	[“encrypt”, “decrypt”]	[128, 192, 256]						Domain 0-128 bits, mod 8.	At least one set of capabilities is required. See Table 10
AES-FF3-1	“1.0”	[“encrypt”, “decrypt”]	[128, 192, 256]							At least one set of capabilities is required. See Table 10
NOTE 1 – keyingOption 2 **SHALL** only be available for decrypt operations. NOTE 2 – AES-CTR implementations must support a payloadLen of 128-bits. For AES-CTR, when values less than 128 are supplied for payloadLen, these lengths refer to the bit sizes supported in the last incomplete block (less than 128 bits) of the cipher or plain text.										

The following grid outlines which properties are **REQUIRED** within the capabilities object array in use for ACVP-AES-FF1 and ACVP-AES-FF3-1.

Table 10 — Format Preserving Encryption Capabilities Object

Property Name	Description	Type	Valid Values	
alphabet	An alphabet the IUT supports for Format Preserving Encryption. Example “0123456789abcdefghijklmnopqrstuvwxyz”.	string	Alphanumeric non repeating characters.	

Property Name	Description	Type	Valid Values	
	Alphabets should be a minimum of two characters, and a maximum of 64 (all numbers and upper and lower case letters, additionally "+" and "/").			
radix	The number base for this capability, should match the number of characters from the alphabet.	integer	2-64	
minLen	The minimum payload length the IUT can support for this alphabet.	integer	2 — maxLen	
maxLen	The maximum payload length the IUT can support for this alphabet.	integer	minLen — variable calculation based on radix and algorithm, see [SP 800-38G].	

5. Test Vectors

The ACVP server provides test vectors to the ACVP client, which are then processed and returned to the ACVP server for validation. A typical ACVP validation test session would require multiple test vector sets to be downloaded and processed by the ACVP client. Each test vector set represents an individual algorithm defined during the capability exchange. This section describes the JSON schema for a test vector set used with Symmetric Block Cipher algorithms.

The test vector set JSON schema is a multi-level hierarchy that contains meta data for the entire vector set as well as individual test vectors to be processed by the ACVP client. The following table describes the JSON elements at the top level of the hierarchy.

Table 11 — Top Level Test Vector JSON Elements

JSON Values	Description	JSON Type
acvVersion	Protocol version identifier	string
vsId	Unique numeric vector set identifier	integer
algorithm	Algorithm defined in the capability exchange	string
mode	Mode defined in the capability exchange	string
revision	Protocol test revision selected	string
testGroups	Array of test groups containing test data, see Section 5.1	array

An example of this would look like this

```
{
  "acvVersion": "version",
  "vsId": 1,
  "algorithm": "Alg1",
  "mode": "Mode1",
  "revision": "Revision1.0",
  "testGroups": [ ... ]
}
```

Figure 20

5.1. Test Groups

Test vector sets **MUST** contain one or more test groups, each sharing similar properties. For instance, all test vectors that use the same key size would be grouped together. The testGroups element at the top level of the test vector JSON object **SHALL** be the array of test groups. The Test Group JSON object **MUST** contain meta-data that applies to all test cases within the group. The following table describes the JSON elements that **MAY** appear from the server in the Test Group JSON object:

Table 12 — Test Group JSON Object

JSON Value	Description	JSON type
tgId	Numeric identifier for the test group, unique across the entire vector set.	integer
testType	The test category type (AFT, MCT or counter). See Section 3 for more information about what these tests do, and how to implement them.	string
direction	The IUT processing direction: encrypt or decrypt	string
ivGen	IV generation method	string
ivGenMode	IV generation method	string
saltGen	Salt generation method	string
keyLen	Length of key in bits to use	integer
keyingOption	The TDES keying option to use	integer
ivLen	Length of IV in bits to use	integer
payloadLen	Length of plaintext or ciphertext in bits to use	integer
aadLen	Length of AAD in bits to use	integer
tagLen	Length of AEAD tag in bits to use	integer
alphabet	Characters representing the alphabet in use for the group. ACVP-AES-FF1 and ACVP-AES-FF3-1 only.	string
radix	The number base in use for the group (should match the number of characters from the alphabet. ACVP-AES-FF1 and ACVP-AES-FF3-1 only.	integer
tweakMode	Indicates the format of the tweak value input for AES-XTS. A value of 'hex' indicates that the test cases in the test group will contain a 'tweakValue' element with a 32-character hexadecimal string for a value. A value of 'number' indicates that the test cases in the test group will contain a 'sequenceNumber' (Data Unit Sequence Number) element with an integer value between 0 and 255.	string
tests	Array of individual test case JSON objects, which are defined in Section 5.2	array of testCase objects

Some properties **MUST** appear in the prompt file from the server for every testGroup object. They are as follows:

- tgId

- direction
- testType
- tests

The other properties **MAY** appear depending on the algorithm selected for the test vector set. The following grid defines the **REQUIRED** properties for each standard block cipher, as well as the valid values a server **MAY** use:

Table 13 — Prompt Test Group Block Cipher Applicability Grid

algorithm	revision	keyLen	keyingOption
ACVP-AES-ECB	"1.0"	128, 192, 256	
ACVP-AES-CBC	"1.0"	128, 192, 256	
ACVP-AES-OFB	"1.0"	128, 192, 256	
ACVP-AES-CFB1	"1.0"	128, 192, 256	
ACVP-AES-CFB8	"1.0"	128, 192, 256	
ACVP-AES-CFB128	"1.0"	128, 192, 256	
ACVP-AES-FF1	"1.0"	128, 192, 256	
ACVP-AES-FF3-1	"1.0"	128, 192, 256	
ACVP-TDES-ECB	"1.0"		1, 2
ACVP-TDES-CBC	"1.0"		1, 2
ACVP-TDES-CBCI	"1.0"		1, 2
ACVP-TDES-CFB1	"1.0"		1, 2
ACVP-TDES-CFB8	"1.0"		1, 2
ACVP-TDES-CFB64	"1.0"		1, 2
ACVP-TDES-CFBP1	"1.0"		1, 2
ACVP-TDES-CFBP8	"1.0"		1, 2
ACVP-TDES-CFBP64	"1.0"		1, 2
ACVP-TDES-OFB	"1.0"		1, 2
ACVP-TDES-OFBI	"1.0"		1, 2

The following grid defines when each property is **REQUIRED** from a server for each authenticated block cipher:

Table 14 — Prompt Test Group Authenticated Block Cipher Applicability Grid

algorithm	revision	keyLen	ivGen	ivGenMode	saltGen	ivLen	payloadLen	aadLen	tagLen	saltLen
ACVP-AES-GCM	"1.0"	128, 192, 256	"internal", "external"	"8.2.1", "8.2.2"		within domain	within domain	within domain	within domain	
ACVP-AES-GCM-SIV	"1.0"	128, 256				96	within domain	within domain	128	
ACVP-AES-XPN	"1.0"	128, 192, 256	"internal", "external"	"8.2.1", "8.2.2"	"internal", "external"	96	within domain	within domain	within domain	96

algorithm	revision	keyLen	ivGen	ivGenMode	saltGen	ivLen	payloadLen	aadLen	tagLen	saltLen
ACVP-AES-CCM	"1.0"	128, 192, 256				within domain	within domain	within domain	within domain	
NOTE – The particular values of a domain are **REQUIRED** to be an integer element of the domain present in the registration used. The ACVP server **MAY** select predetermined or random values with particular features (ex. on a block boundary, or not on a block boundary) within the domain the client provided in the registration.										

The following grid defines when each property is **REQUIRED** from a server for a key-wrap block cipher:

Table 15 — Prompt Test Group Key-Wrap Block Cipher Applicability Grid

algorithm	revision	keyLen	kwCipher	payloadLen
ACVP-AES-KW	"1.0"	128, 192, 256	"cipher", "inverse"	within domain
ACVP-AES-KWP	"1.0"	128, 192, 256	"cipher", "inverse"	within domain
ACVP-TDES-KW	"1.0"		"cipher", "inverse"	within domain
NOTE – The particular values of a domain are **REQUIRED** to be an integer element of the domain present in the registration used. The ACVP server **MAY** select predetermined or random values with particular features (ex. on a block boundary, or not on a block boundary) within the domain the client provided in the registration.				

The following grid defines when each property is **REQUIRED** from a server for the miscellaneous block ciphers:

Table 16 — Prompt Test Group Miscellaneous Block Cipher Applicability Grid

algorithm	revision	keyLen	keyingOption	incremental	overflow	tweakMode	payloadLen
ACVP-AES-CBC-CS1	"1.0"	128, 192, 256					
ACVP-AES-CBC-CS2	"1.0"	128, 192, 256					
ACVP-AES-CBC-CS3	"1.0"	128, 192, 256					
ACVP-AES-CTR	"1.0"	128, 192, 256		true, false	true, false		
ACVP-AES-XTS	"1.0"	128, 256				"hex", "number"	within domain
ACVP-AES-XTS	"2.0"	128, 256				"hex", "number"	

algorithm	revision	keyLen	keyingOption	incremental	overflow	tweakMode	payloadLen
ACVP-TDES-CTR	"1.0"		1, 2	true, false	true, false		
ACVP-AES-FF1	"1.0"	128, 192, 256					
ACVP-AES-FF3-1	"1.0"	128, 192, 256					
NOTE – The particular values of a domain are **REQUIRED** to be an integer element of the domain present in the registration used. The ACVP server **MAY** select predetermined or random values with particular features (ex. on a block boundary, or not on a block boundary) within the domain the client provided in the registration.							

5.2. Test Cases

Each test group **SHALL** contain an array of one or more test cases. Each test case is a JSON object that represents a single case to be processed by the ACVP client. The following table describes the JSON elements for each test case.

Table 17 — Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set.	integer
key	Encryption key to use for AES	string (hex)
key1, key2, key3	Encryption keys to use for TDES	string (hex)
iv	IV to use	string (hex)
tweak	Tweak used to form an IV for AES-FF1 and AES-FF3-1	string (hex)
tweakLen	Length of the tweak for AES-FF1 and AES-FF3-1	integer
tweakValue	Tweak value used to form an IV for AES-XTS when the tweakMode for the group is 'hex'. A 32-character hexadecimal string.	string (hex)
sequenceNumber	(Data Unit Sequence Number) Integer used to form an IV for AES-XTS when the tweakMode for the group is 'number'. An integer between 0 and 255.	integer
salt	The salt to use in AES-XPN (required for AES-XPN only)	string (hex)

JSON Value	Description	JSON type
pt	Plaintext to use	string (hex)
ct	Ciphertext to use	string (hex)
payloadLen	The length of the provided Plaintext or Ciphertext in bits. Only the most significant 'payloadLen' bits will be used.	integer
dataUnitLen	Length of the data unit in bits for ACVP-AES-XTS	integer
aad	AAD to use for AEAD algorithms	string (hex)
tag	Tag to use for AEAD algorithms	string (hex)
NOTE – The applicability of each test case property is dependent on the test group and test vector (algorithm) properties. Each test type within the test group requires specific operations to be performed and thus specific data returned to the server. Consult Section 3 for more information. The tcId property **MUST** appear within every test case sent to and from the server.		

The following grid identifies the algorithms whose test case JSON objects will contain the 'payloadLen' property and the valid values a servery **MAY** use.

Table 18 — Applicability of the 'payloadLen' Property

Algorithm	Revision	Applicability	payloadLen
ACVP-AES-CBC-CS1	"1.0"	for all test cases	within domain (see Table 9)
ACVP-AES-CBC-CS2	"1.0"	for all test cases	within domain (see Table 9)
ACVP-AES-CBC-CS3	"1.0"	for all test cases	within domain (see Table 9)
ACVP-AES-CFB1	"1.0"	for all test cases	1
ACVP-AES-CTR	"1.0"	for the test cases of some test groups, but not others	within domain (see Table 9)
ACVP-AES-XTS	"2.0"	for all test cases	within domain (see Table 8)
NOTE – Table 18 identifies the algorithms for which the 'payloadLen' property will be present as part of the test case JSON objects. Please also be aware that other algorithms include the 'payloadLen' property as part of the test group JSON object. For more information, see Table 14, Table 15, and Table 16.			

6. Test Vector Responses

After the ACVP client downloads and processes a vector set, it **SHALL** send the response vectors back to the ACVP server within the alloted timeframe. The following table describes the JSON object that represents a vector set response.

Table 19 — Vector Set Response JSON Object

JSON Value	Description	JSON type
acvVersion	Protocol version identifier	string
vsId	Unique numeric identifier for the vector set	integer
testGroups	Array of JSON objects that represent each test vector group. See Table 20	array of testGroup objects

The testGroup Response section is used to organize the ACVP client response in a similar manner to how it receives vectors. Several algorithms **SHALL** require the client to send back group level properties in its response. This structure helps accommodate that.

Table 20 — Vector Set Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
tests	The tests associated to the group specified in tgId	array of testCase objects

Each test case is a JSON object that represents a single test object to be processed by the ACVP client. The following table describes the JSON elements for each test case object.

Table 21 — Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set.	integer
pt	The IUT's pt response to a decrypt test	string (hex)
ct	The IUT's ct response to an encrypt test	string (hex)
testPassed	Some test cases included with decrypt operations in AES-GCM, AES-CCM, AES-XPN, AES-KW, AES-KWP, and TDES-KW will have expected failures.	boolean

JSON Value	Description	JSON type
resultsArray	Array of JSON objects that represent each iteration of a Monte Carlo Test. Each iteration will contain the key(s), pt, ct and iv	array of objects containing pt, ct and iv (except for ECB mode)
NOTE – The tcId **MUST** be included in every test case object sent between the client and the server.		

7. Security Considerations

There are no additional security considerations outside of those outlined in the ACVP document.

Appendix A — References

S. Bradner (March 1997) *Key words for use in RFCs to Indicate Requirement Levels* (Internet Engineering Task Force), BCP 14, March 1997. RFC 2119. RFC RFC2119. DOI 10.17487/RFC2119. https://www.rfc-editor.org/info/rfc2119.

B. Leiba (May 2017) *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words* (Internet Engineering Task Force), BCP 14, May 2017. RFC 8174. RFC RFC8174. DOI 10.17487/RFC8174. https://www.rfc-editor.org/info/rfc8174.

National Institute of Standards and Technology (November 2001) *Advanced Encryption Standard (AES)* (Gaithersburg, MD), November 2001. FIPS 197. https://doi.org/10.6028/NIST.FIPS.197.

Morris J. Dworkin (December 2001) *Recommendation for Block Cipher Modes of Operation—Methods and Techniques* (Gaithersburg, MD), December 2001. SP 800-38A. https://doi.org/10.6028/NIST.SP.800-38A.

Morris J. Dworkin (December 2001) *Recommendation for Block Cipher Modes of Operation—Methods and Techniques* (Gaithersburg, MD), December 2001. SP 800-38A. https://doi.org/10.6028/NIST.SP.800-38A.

Morris J. Dworkin (May 2004 (updated July 2007)) *Recommendation for Block Cipher Modes of Operation—the CCM Mode for Authentication and Confidentiality* (Gaithersburg, MD), May 2004 (updated July 2007). SP 800-38C. https://doi.org/10.6028/NIST.SP.800-38C.

Morris J. Dworkin (November 2007) *Recommendation for Block Cipher Modes of Operation—Galois/Counter Mode (GCM) and GMAC* (Gaithersburg, MD), November 2007. SP 800-38D. https://doi.org/10.6028/NIST.SP.800-38D.

Morris J. Dworkin (January 2010) *Recommendation for Block Cipher Modes of Operation—the XTS-AES Mode for Confidentiality on Storage Devices* (Gaithersburg, MD), January 2010. SP 800-38E. https://doi.org/10.6028/NIST.SP.800-38E.

Morris J. Dworkin (December 2012) *Recommendation for Block Cipher Modes of Operation—Methods for Key Wrapping* (Gaithersburg, MD), December 2012. SP 800-38F. https://doi.org/10.6028/NIST.SP.800-38F.

Morris J. Dworkin (March 2016) *Recommendation for Block Cipher Modes of Operation—Methods for Format-Preserving Encryption* (Gaithersburg, MD), March 2016. SP 800-38G. https://doi.org/10.6028/NIST.SP.800-38G.

Elaine B. Barker, Dr. Nicky Mouha (November 2017) *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher* (Gaithersburg, MD), November 2017. SP 800-67 Rev. 2. https://doi.org/10.6028/NIST.SP.800-67r2.

R. Housley (January 2004) *Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)* (Internet Engineering Task Force), RFC 3686, January 2004. RFC 3686. RFC RFC3686. DOI 10.17487/RFC3686. https://www.rfc-editor.org/info/rfc3686.

Fussell B, Vassilev A, Booth H, Celi C, Hammett R (July 01, 2019) *Automatic Cryptographic Validation Protocol* (National Institute of Standards and Technology, Gaithersburg, MD), July 01, 2019.

Gueron S, Langley A, Lindell Y (2018) *AES-GCM-SIV Nonce Misuse-Resistant Authenticated Encryption* (University of Haifa and Amazon Web Services, Google LLC, and Bar Ilan University), 2018. AES-GCM-SIV.

ECMA-368 High Rate Ultra Wideband PHY and MAC Standard. ECMA. https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-368.pdf.

Thompson T (March 2008) *IEEE 1619-2007—IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices*, March 2008. IEEE 1619-2007. https://standards.ieee.org/standard/1619-2007.html.

Appendix B — Example Capabilities JSON Object

The following is a example JSON object advertising support for all block ciphers.

```
[{
    "acvVersion": "{acvp-version}"
  }, {
    "algorithm": "ACVP-AES-GCM",
    "revision": "1.0",
    "prereqVals": [{
        "algorithm": "ACVP-AES-ECB",
        "valValue": "123456"
      },
      {
        "algorithm": "DRBG",
        "valValue": "123456"
      }
    ],
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "ivGen": "internal",
    "ivGenMode": "8.2.2",
    "keyLen": [
      128,
      192,
      256
    ],
    "tagLen": [
      96,
      128
    ],
    "ivLen": [
      96
    ],
    "payloadLen": [
      0,
      256
    ],
    "aadLen": [
      128,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-ECB",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-CBC",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-CBC-CS1",
    "revision": "1.0",
    "payloadLen": [{
      "min": 128,
      "max": 65536,
      "increment": 1
    }],
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-CBC-CS2",
    "revision": "1.0",
    "payloadLen": [{
      "min": 128,
      "max": 65536,
      "increment": 1
    }],
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-CBC-CS3",
    "revision": "1.0",
    "payloadLen": [{
      "min": 128,
      "max": 65536,
      "increment": 1
    }],
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-CFB8",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-CFB128",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-OFB",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-XPN",
    "revision": "1.0",
    "prereqVals": [{
        "algorithm": "ACVP-AES-ECB",
        "valValue": "123456"
      },
      {
        "algorithm": "DRBG",
        "valValue": "123456"
      }
    ],
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "ivGen": "internal",
    "ivGenMode": "8.2.2",
    "saltGen": "internal",
    "keyLen": [
      128,
      192,
      256
    ],
    "tagLen": [
      96,
      128
    ],
    "payloadLen": [
      0,
      128
    ],
    "aadLen": [
      120,
      128
    ]
  },
  {
    "algorithm": "ACVP-AES-CTR",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ],
    "payloadLen": [
      128
    ],
    "incrementalCounter": true,
    "overflowCounter": false
  },
  {
    "algorithm": "ACVP-AES-CTR",
    "revision": "1.0",
    "conformances": ["RFC3686"],
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
        256
      ],
      "payloadLen": [
        128
      ],
      "incrementalCounter": true,
      "overflowCounter": false,
      "ivGenMode": "external"
    },
    {
      "algorithm": "ACVP-AES-CCM",
      "revision": "1.0",
      "prereqVals": [{
        "algorithm": "ACVP-AES-ECB",
        "valValue": "same"
      }],
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyLen": [
        128,
        192,
        256
      ],
      "tagLen": [
        128
      ],
      "ivLen": [
        56
      ],
      "payloadLen": [
        0,
        256
      ],
      "aadLen": [
        0,
        65536
      ]
    },
    {
      "algorithm": "ACVP-AES-CFB1",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ]
  },
  {
    "algorithm": "ACVP-AES-KW",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "kwCipher": [
      "cipher"
    ],
    "keyLen": [
      128,
      192,
      256
    ],
    "payloadLen": [
      512,
      192,
      128
    ]
  },
  {
    "algorithm": "ACVP-AES-KWP",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "kwCipher": [
      "cipher"
    ],
    "keyLen": [
      128,
      192,
      256
    ],
    "payloadLen": [
      8,
      32,
      96,
      808
    ]
  },
  {
    "algorithm": "ACVP-AES-FF1",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ],
    "tweakLen": [{
      "min": 0,
      "max": 128,
      "increment": 8
    }],
    "capabilities": [{
        "alphabet": "0123456789",
        "radix": 10,
        "minLen": 10,
        "maxLen": 56
      },
      {
        "alphabet": "abcdefghijklmnopqrstuvwxyz",
        "radix": 26,
        "minLen": 10,
        "maxLen": 40
      },
      {
        "alphabet":
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/",
        "radix": 64,
        "minLen": 10,
        "maxLen": 28
      }
    ]
  },
  {
    "algorithm": "ACVP-AES-FF3-1",
    "revision": "1.0",
    "conformances": [],
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      192,
      256
    ],
    "tweakLen": [{
      "min": 0,
      "max": 128,
      "increment": 8
    }],
    "capabilities": [{
        "alphabet": "0123456789",
        "radix": 10,
        "minLen": 10,
        "maxLen": 56
      },
      {
        "alphabet": "abcdefghijklmnopqrstuvwxyz",
        "radix": 26,
        "minLen": 10,
        "maxLen": 40
      },
      {
        "alphabet":
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/",
        "radix": 64,
        "minLen": 10,
        "maxLen": 28
      }
    ]
  },
  {
    "algorithm": "ACVP-AES-XTS",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      256
    ],
    "payloadLen": [
      65536
    ],
    "tweakMode": [
      "hex",
      "number"
    ]
  },
  {
    "algorithm": "ACVP-AES-XTS",
    "revision": "2.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyLen": [
      128,
      256
    ],
    "payloadLen": [
      65536
    ],
    "tweakMode": [
      "hex",
      "number"
    ],
    "dataUnitLen": [
      1024, 4096
    ]
  },
  {
    "algorithm": "ACVP-TDES-ECB",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyingOption": [
      1
    ]
  },
  {
    "algorithm": "ACVP-TDES-CBC",
    "revision": "1.0",
    "direction": [
      "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-CBCI",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-OFB",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-OFBI",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-CFB64",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-CFB8",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-CFB1",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-CFBP64",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
      ]
    },
    {
      "algorithm": "ACVP-TDES-CFBP8",
      "revision": "1.0",
      "direction": [
        "encrypt",
        "decrypt"
      ],
      "keyingOption": [
        1
    ]
  },
  {
    "algorithm": "ACVP-TDES-CFBP1",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyingOption": [
      1
    ]
  },
  {
    "algorithm": "ACVP-TDES-CTR",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "keyingOption": [
      1
    ],
    "payloadLen": [
      64
    ],
    "performCounterTests": false
  },
  {
    "algorithm": "ACVP-TDES-KW",
    "revision": "1.0",
    "direction": [
      "encrypt",
      "decrypt"
    ],
    "kwCipher": [
      "cipher"
    ],
    "payloadLen": [
      512,
      192,
      128
    ]

  }
```

]

Figure B-1

Appendix C — Example Vector Set Request/Responses JSON Object

The following sections provide examples of the JSON objects for each of the AES algorithms. Examples will reflect what testTypes are supported by each algorithm, ie AFT, MCT or counter. MCT examples have only 2 iterations shown for brevity.

The following shows AES-GCM AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2055,
  "algorithm": "ACVP-AES-GCM",
  "revision": "1.0",
  "testGroups": [{
            "tgId": 1,
            "testType": "AFT",
            "direction": "encrypt",
            "keyLen": 128,
            "ivLen": 96,
            "ivGen": "external",
            "ivGenMode": "8.2.2",
            "payloadLen": 0,
            "aadLen": 0,
            "tagLen": 32,
            "tests": [
                {
                    "tcId": 1,
                    "plainText": "",
                    "key": "10B8D4C9658590A...",
                    "aad": "",
                    "iv": "3D026F3D590BF1A7..."
                },
                {
                    "tcId": 2,
                    "plainText": "",
                    "key": "934865822A3ECCB...",
                    "aad": "",
                    "iv": "273F3B30341C779E..."
                }
      ]
    },
    {
            "tgId": 19,
            "testType": "AFT",
            "direction": "decrypt",
            "keyLen": 128,
            "ivLen": 96,
            "ivGen": "external",
            "ivGenMode": "8.2.2",
            "payloadLen": 0,
            "aadLen": 120,
            "tagLen": 32,
            "tests": [
                {
                    "tcId": 271,
                    "key": "88AB5441AE2...",
                    "aad": "4E956EF528D...",
                    "iv": "810628011BB0...",
                    "cipherText": "",
                    "tag": "1180FD89"
                },
                {
                    "tcId": 272,
                    "key": "9149BE47FAEB...",
                    "aad": "938A8FA71324...",
                    "iv": "FF6B72FF25B55...",
                    "cipherText": "",
                    "tag": "6C7528F0"
                }
      ]
    }
  ]
}]
```

Figure C-1

The following shows AES-GCM AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2055,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "iv": "01020304F966B8...",
          "ct": "",
          "tag": "427F668E58F56..."
        },
        {
          "tcId": 2,
          "iv": "01020304C2855...",
          "ct": "",
          "tag": "D95BD66F7789..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 902,
          "pt": "763BF..."
        },
        {
          "tcId": 903,
          "testPassed": false
        }
      ]
    }
  ]
}]
```

Figure C-2

The following shows AES-CCM AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2061,
  "algorithm": "ACVP-AES-CCM",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "direction": "encrypt",
    "testType": "AFT",
    "ivLen": 56,
    "payloadLen": 256,
    "aadLen": 0,
    "tagLen": 128,
    "keyLen": 128,
    "tests": [{
      "tcId": 1,
      "pt": "361445511E0BD3E94E3...",
      "key": "7DB9E755181E4160C6...",
      "iv": "1C53ECD62BBED5",
      "aad": ""
    }, {
      "tcId": 2,
      "pt": "735CE37215A91074DBF...",
      "key": "7DB9E755181E4160C6...",
      "iv": "1C53ECD62BBED5",
      "aad": ""
    }]
  }, {
    "tgId": 2,
    "direction": "decrypt",
    "testType": "AFT",
    "ivLen": 56,
    "payloadLen": 0,
    "aadLen": 0,
    "tagLen": 128,
    "keyLen": 128,
    "tests": [{
      "tcId": 181,
      "ct": "533427D475EBAC3FE5...",
      "key": "A8B7C7A69E5AB940B...",
      "iv": "1BD5816AF5BB9F",
      "aad": ""
    }, {
      "tcId": 182,
      "ct": "6B774BB2D20A8A23A1...",
      "key": "A8B7C7A69E5AB940B...",
      "iv": "8140308B19BCE8",
      "aad": ""
    }]
  }]
}]
```

Figure C-3

The following shows AES-CCM AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
    "vsId": 2061,
    "testGroups": [{
            "tgId": 1,
            "tests": [{
                    "tcId": 1,
                    "ct": "C8AB4A739E1..."
                },
                {
                    "tcId": 2,
                    "ct": "8DE3EC5095B..."
                }
            ]
        },
        {
            "tgId": 2,
            "tests": [{
                    "tcId": 181,
                    "testPassed": false
                },
                {
                    "tcId": 182,
                    "pt": ""
                }
            ]
        }
    ]
}]
```

Figure C-4

The following shows AES-CBC AFT and MCT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2057,
  "algorithm": "ACVP-AES-CBC",
    "revision": "1.0",
  "testGroups": [{
      "tgId": 1,
      "direction": "encrypt",
      "testType": "AFT",
      "keyLen": 128,
      "tests": [{
        "tcId": 1,
        "iv": "00C8F42C5B5...",
        "key": "7F9863BCD5...",
        "pt": "97549D671FA..."
      }, {
        "tcId": 2,
        "iv": "CE6747E918F...",
        "key": "25F73DBAF4...",
        "pt": "D3A0AA732D7..."
      }]
    }, {
      "tgId": 2,
        "direction": "decrypt",
        "testType": "AFT",
        "keyLen": 128,
        "tests": [{
          "tcId": 31,
          "iv": "D498F4F8462...",
          "key": "77D563ACE1...",
          "ct": "78256FA155F..."
        }, {
          "tcId": 32,
          "iv": "AB99A939B688...",
          "key": "0569B0C6DB3...",
          "ct": "EEBF23A65E83..."
        }]
      },
      {
        "tgId": 3,
        "direction": "encrypt",
        "testType": "MCT",
        "keyLen": 256,
        "tests": [{
          "tcId": 63,
          "iv": "057FB7EEDE1EBF40...",
          "key": "E5E2E9F088E2C06...",
          "pt": "6DA46A0AADB59615..."
        }]
      }, {
        "tgId": 4,
        "direction": "decrypt",
        "testType": "MCT",
        "keyLen": 128,
        "tests": [{
          "tcId": 64,
          "iv": "FD5EDEC164E504D6...",
          "key": "F7439EAC671FC4B...",
          "ct": "37ECE2FF3F391D8C..."
        }]
      }
    ]
}]
```

Figure C-5

The following shows AES-CBC AFT and MCT responses.

```
[{
  "acvVersion": "{acvp-version}"
```

```
},{
  "vsId": 2057,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "DD95E867DFCFCC..."
        },
        {
          "tcId": 2,
          "ct": "540954F0016D40..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "F7251EA3C68FE..."
        },
        {
          "tcId": 32,
          "pt": "CEC14A7B465A3..."
        }
      ]
    },
    {
      "tgId": 3,
      "tests": [{
        "tcId": 63,
        "resultsArray": [{
            "key": "E5E2...",
            "iv": "057FB...",
            "pt": "6DA46...",
            "ct": "3E794..."
          },
          {
            "key": "DE31...",
            "iv": "3E794...",
            "pt": "3BD32...",
            "ct": "9236D..."
          }
        ]
      }]
    },
    {
      "tgId": 1,
      "tests": [{
        "tcId": 64,
        "resultsArray": [{
            "key": "F743...",
            "iv": "FD5ED...",
            "ct": "37ECE...",
            "pt": "52FC3..."
          },
          {
            "key": "A5BF...",
            "iv": "52FC3...",
            "ct": "4400F...",
            "pt": "66204..."
          }
        ]
      }]
    }
  ]
}]
```

Figure C-6

The following shows AES-CBC-CS1 AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2058,
  "algorithm": "ACVP-AES-CBC-CS1",
    "revision": "1.0",
  "testGroups": [{
      "tgId": 1,
      "direction": "encrypt",
      "testType": "AFT",
      "keyLen": 128,
      "tests": [{
        "tcId": 1,
        "iv": "1216A541024...",
        "key": "A6A8346C47...",
        "pt": "71AC206DD0A...",
        "payloadLen": 512
      }, {
        "tcId": 2,
        "iv": "9A6A276AB96...",
        "key": "7CDAE90854...",
        "pt": "6D4AEE90179...",
        "payloadLen": 178
      }]
    }, {
      "tgId": 2,
      "direction": "decrypt",
      "testType": "AFT",
      "keyLen": 128,
      "tests": [{
        "tcId": 31,
        "iv": "908543E2646...",
        "key": "CB12AAFA25B...",
        "ct": "AB99A939B688...",
        "payloadLen": 378
      }, {
        "tcId": 32,
        "iv": "AB99A939B688...",
        "key": "0569B0C6DB3...",
        "ct": "DD14A9A9A916A...",
        "payloadLen": 471
      }]
    }
  ]
}]
```

Figure C-7

The following shows AES-CBC-CS1 AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2057,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "E25DC48F39E4DA..."
        },
        {
          "tcId": 2,
          "ct": "360D25D820C3BA..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "33346D02A070A..."
        },
        {
          "tcId": 32,
          "pt": "8F52D6E73783A..."
        }
      ]
    }
  ]
}]
```

Figure C-8

The following shows AES-CBC-CS2 AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2058,
  "algorithm": "ACVP-AES-CBC-CS2",
    "revision": "1.0",
  "testGroups": [{
      "tgId": 1,
      "direction": "encrypt",
      "testType": "AFT",
      "keyLen": 128,
      "tests": [{
        "tcId": 1,
        "iv": "1216A541024...",
        "key": "A6A8346C47...",
        "pt": "71AC206DD0A...",
        "payloadLen": 512
      }, {
        "tcId": 2,
        "iv": "9A6A276AB96...",
        "key": "7CDAE90854...",
        "pt": "6D4AEE90179...",
        "payloadLen": 178
      }]
    }, {
      "tgId": 2,
      "direction": "decrypt",
      "testType": "AFT",
      "keyLen": 128,
      "tests": [{
        "tcId": 31,
        "iv": "908543E2646...",
        "key": "CB12AAFA25B...",
        "ct": "AB99A939B688...",
        "payloadLen": 378
      }, {
        "tcId": 32,
        "iv": "AB99A939B688...",
        "key": "0569B0C6DB3...",
        "ct": "DD14A9A9A916A...",
        "payloadLen": 471
      }]
    }
  ]
}]
```

Figure C-9

The following shows AES-CBC-CS2 AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2057,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "E25DC48F39E4DA..."
        },
        {
          "tcId": 2,
          "ct": "360D25D820C3BA..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "33346D02A070A..."
        },
        {
          "tcId": 32,
          "pt": "8F52D6E73783A..."
        }
      ]
    }
  ]
}]
```

Figure C-10

The following shows AES-CBC-CS3 AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2058,
  "algorithm": "ACVP-AES-CBC-CS3",
    "revision": "1.0",
  "testGroups": [{
      "tgId": 1,
      "direction": "encrypt",
      "testType": "AFT",
      "keyLen": 128,
      "tests": [{
        "tcId": 1,
        "iv": "1216A541024...",
        "key": "A6A8346C47...",
        "pt": "71AC206DD0A...",
        "payloadLen": 512
      }, {
        "tcId": 2,
        "iv": "9A6A276AB96...",
        "key": "7CDAE90854...",
        "pt": "6D4AEE90179...",
        "payloadLen": 178
      }]
    }, {
      "tgId": 2,
      "direction": "decrypt",
      "testType": "AFT",
      "keyLen": 128,
      "tests": [{
        "tcId": 31,
        "iv": "908543E2646...",
        "key": "CB12AAFA25B...",
        "ct": "AB99A939B688...",
        "payloadLen": 378
      }, {
        "tcId": 32,
        "iv": "AB99A939B688...",
        "key": "0569B0C6DB3...",
        "ct": "DD14A9A9A916A...",
        "payloadLen": 471
      }]
    }
  ]
}]
```

Figure C-11

The following shows AES-CBC-CS3 AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2057,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "E25DC48F39E4DA..."
        },
        {
          "tcId": 2,
          "ct": "360D25D820C3BA..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "33346D02A070A..."
        },
        {
          "tcId": 32,
          "pt": "8F52D6E73783A..."
        }
      ]
    }
  ]
}]
```

Figure C-12

The following shows AES-ECB AFT and MCT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2056,
  "algorithm": "ACVP-AES-ECB",
    "revision": "1.0",
  "testGroups": [{
      "tgId": 1,
            "testType": "AFT",
            "direction": "encrypt",
            "keylen": 128,
            "tests": [
                {
                    "tcId": 1,
                    "plainText": "F34481E...",
                    "key": "0000000000000..."
                },
                {
                    "tcId": 2,
                    "plainText": "9798C46...",
                    "key": "0000000000000..."
                }
      ]
    },
    {
            "tgId": 25,
            "testType": "AFT",
            "direction": "encrypt",
            "keylen": 128,
            "tests": [
                {
                    "tcId": 2079,
                    "plainText": "1C46FA6...",
                    "key": "18D3248D32630..."
                },
                {
                    "tcId": 2080,
                    "plainText": "5AC1B2D...",
                    "key": "26007B74016FA..."
                }
      ]
    },
        {
        "tgId": 31,
        "testType": "MCT",
        "direction": "encrypt",
        "keylen": 128,
        "tests": [
                {
                    "tcId": 2139,
                    "key": "9489F6FFA4A74...",
              "pt": "2D984D2F1FC178..."
                }
            ]
        },
        {
        "tgId": 34,
        "testType": "MCT",
        "direction": "decrypt",
        "keylen": 128,
        "tests": [
                {
                    "tcId": 2142
                    "key": "9489F6FFA4A74...",
              "ct": "2D984D2F1FC178..."
                }
            ]
        }
  ]
}]
```

Figure C-13

The following shows AES-ECB AFT and MCT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2056,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "43FB8A36F168E3..."
        },
        {
          "tcId": 2,
          "ct": "27549D65BE8056..."
        }
      ]
    },
    {
      "tgId": 1,
      "tests": [{
          "tcId": 31,
          "pt": "F7F42B062BD643..."
        },
        {
          "tcId": 32,
          "pt": "EAF9AAA67B6C0E..."
        }
      ]
    },
    {
      "tgId": 3,
      "tests": [{
        "tcId": 61,
        "resultsArray": [{
            "key": "A4A8255E7...",
            "pt": "B3B8F494D0...",
            "ct": "619D5B0921..."
          },
          {
            "key": "C5357E575...",
            "pt": "619D5B0921...",
            "ct": "28CF1C5DD2..."
          }
        ]
      }]
    },
    {
      "tgId": 4,
      "tests": [{
        "tcId": 64,
        "resultsArray": [{
            "key": "4D3BE577E...",
            "ct": "0FE92E22BA...",
            "pt": "73ED187BFE..."
          },
          {
            "key": "3ED6FD0C1...",
            "ct": "73ED187BFE...",
            "pt": "59550A36E1..."
          }
        ]
      }]
    }
  ]
}]
```

Figure C-14

The following shows AES-OFB AFT and MCT request vectors.

```
[{
    "acvVersion": "{acvp-version}",
},{
  "vsId": 2060,
  "algorithm": "ACVP-AES-OFB",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "direction": "encrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 1,
      "iv": "0F24B3F7808F292BC39128...",
      "key": "8ECE26B1880C4B1F0A59E...",
      "pt": "A8EF19C7182527C8CBBEE1..."
    }, {
      "tcId": 2,
      "iv": "1D1CC64F9F004192B6BE35...",
      "key": "054240C952C99D5B6E387224F...",
      "pt": "EBFA3F5F990B678AA884FB..."
    }]
  }, {
    "tgId": 2,
    "direction": "decrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 31,
      "iv": "A5F67A6CB0238A5DFB166...",
      "key": "A3988AC61E9FB4820876...",
      "ct": "CF6F24E68CEC8B97CB88D..."
    }, {
      "tcId": 32,
      "iv": "4098786D4EF05639B5A20...",
      "key": "5D22EAF883FB2B1847BF...",
      "ct": "7203926F1210401F566E0..."
    }]
  }, {
    "tgId": 3,
    "direction": "encrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 61,
      "iv": "39F33D19A09AAFD200D4C...",
      "key": "190316BF21DE21E96FCF...",
      "pt": "E4D7F490829710CADFD67..."
    }]
  }, {
    "tgId": 4,
    "direction": "decrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 64,
      "iv": "1915C8A7AFEBB26AAE97C...",
      "key": "9489F6FFA4A7480D5B34...",
      "ct": "2D984D2F1FC178CAB247F..."
    }]
  }]
}]
```

Figure C-15

The following shows AES-OFB AFT and MCT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2060,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "B5D16C4219AC38..."
        },
        {
          "tcId": 2,
          "ct": "B85AF8646842A9..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "0863AB3A0CA17C..."
        },
        {
          "tcId": 32,
            "pt": "BF69D1BE04D013..."
          }
        ]
      },
      {
        "tgId": 3,
        "tests": [{
          "tcId": 61,
          "resultsArray": [{
              "key": "190316BF...",
              "iv": "39F33D19A...",
              "pt": "E4D7F4908...",
              "ct": "F55626877..."
            },
            {
              "key": "EC553038...",
              "iv": "F55626877...",
              "pt": "A04BCACFF...",
              "ct": "1EAA7DE30..."
            }
          ]
        }]
      },
      {
        "tgId": 4,
        "tests": [{
          "tcId": 64,
          "resultsArray": [{
              "key": "9489F6FF...",
              "iv": "1915C8A7A...",
              "ct": "2D984D2F1...",
              "pt": "0FE5765E5..."
            },
            {
              "key": "9B6C80A1...",
              "iv": "0FE5765E5...",
              "ct": "F29F68E2E...",
              "pt": "39AC0B63E..."
            }
          ]
        }]
      }
    ]
}]
```

Figure C-16

The following shows AES-CFB1 AFT and MCT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2062,
  "algorithm": "ACVP-AES-CFB1",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "direction": "encrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 67,
      "iv": "F34481EC3CC627BACD5DC3...",
      "key": "000000000000000000000...",
      "pt": "00",
      "payloadLen": 1
    }, {
      "tcId": 68,
      "iv": "9798C4640BAD75C7C3227D...",
      "key": "000000000000000000000...",
      "pt": "00",
      "payloadLen": 1
    }]
  }, {
    "tgId": 2,
    "direction": "decrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 31,
      "iv": "C74388BA333118CDBDF578...",
      "key": "8DE5E0586C4EA40FC36C0...",
      "ct": "80",
      "payloadLen": 1
    }, {
      "tcId": 32,
      "iv": "0B1B558F3AF46F2E6AB29D...",
      "key": "E52350E8E8EE950A3C2E3...",
      "ct": "80",
      "payloadLen": 1
    }]
  }, {
    "tgId": 3,
    "direction": "encrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 61,
      "iv": "D4A4A028EEA3BCA708A31E...",
      "key": "A3B254EAB3B0C8C60EF6A...",
      "pt": "80",
      "payloadLen": 1
    }]
  }, {
    "tgId": 4,
    "direction": "decrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 64,
      "iv": "75BEE06DEC8A99EC0C7E7F...",
      "key": "7C87174CB990272D0F2F2...",
      "ct": "00",
      "payloadLen": 1
    }]
  }]
}]
```

Figure C-17

The following shows AES-CFB1 AFT and MCT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2062,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 67,
          "ct": "00"
        },
        {
          "tcId": 68,
          "ct": "80"
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "00"
        },
        {
          "tcId": 32,
          "pt": "80"
        }
      ]
    },
    {
      "tgId": 3,
      "tests": [{
        "tcId": 61
        "resultsArray": [{
            "key": "A3B254EAB...",
            "iv": "D4A4A028EE...",
            "pt": "80",
            "ct": "00"
          },
          {
            "key": "8FFC23126...",
            "iv": "2C4E77F8D0...",
            "pt": "00",
            "ct": "00"
          }
        ]
      }]
    }, {
      "tgId": 4,
      "tests": [{
        "tcId": 64
        "resultsArray": [{
            "key": "7C87174CB...",
            "iv": "75BEE06DEC...",
            "ct": "00",
            "pt": "00"
          },
          {
            "key": "4B2492A3F...",
            "iv": "37A385EF42...",
            "ct": "80",
            "pt": "80"
          }
        ]
      }]
```

```
    }
  ]
}]
```

Figure C-18

The following shows AES-CFB8 AFT and MCT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2058,
  "algorithm": "ACVP-AES-CFB8",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "direction": "encrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 1,
      "iv": "4EBD4CE189E6DA65026C2A...",
      "key": "5FA02465F28B76C441C7B...",
      "pt": "AF5E"
    }, {
      "tcId": 2,
      "iv": "9A8017353E953B5AEC4D78...",
      "key": "538EB5E1CBFEA61CC6B3D...",
      "pt": "6ED3759B"
    }]
  }, {
    "tgId": 2,
    "direction": "decrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 31,
      "iv": "1808A0F308838AA6F9F703...",
      "key": "DB7FFD9166E4A5BACB022...",
      "ct": "41DA"
    }, {
      "tcId": 32,
      "iv": "4D75785D44B1B247788186...",
      "key": "7201F5CC867A8DCE044DB...",
      "ct": "E267BC1B"
    }]
  }, {
    "tgId": 3,
    "direction": "encrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 61,
      "iv": "4B8F7DCCAD48776C746B79...",
      "key": "FD0B5848870C7431179EB...",
      "pt": "AD"
    }]
  }, {
    "tgId": 4,
    "direction": "decrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 64,
      "iv": "5D2080050855970CE15DC1...",
      "key": "EA378F16FF6144EF58E67...",
      "ct": "83"
    }]
  }]
}]
```

Figure C-19

The following shows AES-CFB8 AFT and MCT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2058,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "181B"
        },
        {
          "tcId": 2,
          "ct": "DFF540F0"
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "DA19"
        },
        {
          "tcId": 32,
          "pt": "B2133D11"
        }
      ]
    },
    {
      "tgId": 3,
      "tests": [{
        "tcId": 61,
        "resultsArray": [{
            "key": "FD0B58488...",
            "iv": "4B8F7DCCAD...",
            "pt": "AD",
            "ct": "3A"
          },
          {
            "key": "6B96D9FD0...",
            "iv": "969D81B585...",
            "pt": "2F",
            "ct": "BD"
          }
        ]
      }]
    },
    {
      "tgId": 4,
      "tests": [{
        "tcId": 64,
        "resultsArray": [{
            "key": "EA378F16F...",
            "iv": "5D20800508...",
            "ct": "83",
            "pt": "E6"
          },
          {
            "key": "31A0B0001A...",
            "iv": "DB973F16E5D...",
            "ct": "24",
            "pt": "0A"
          }
        ]
      }]
    }
  ]
}]
```

Figure C-20

The following shows AES-CFB128 AFT and MCT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2059,
  "algorithm": "ACVP-AES-CFB128",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "direction": "encrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 1,
      "iv": "24AD71C9734E64B8AC458...",
      "key": "55B2490AD74A470F5CFE...",
      "pt": "FE9C6B296C58324FE8B48..."
    }, {
      "tcId": 2,
      "iv": "C0042889D189B508C5B88...",
      "key": "AB383065E16B17306B50...",
      "pt": "19F109316F7F740BD48FF..."
    }]
  }, {
    "tgId": 2,
    "direction": "decrypt",
    "testType": "AFT",
    "keyLen": 128,
    "tests": [{
      "tcId": 31,
      "iv": "40619E2F346B02D49BCEE...",
      "key": "744F5B5D7813974E0DE2...",
      "ct": "5B12E9B418F720C344698..."
    }, {
      "tcId": 32,
      "iv": "D571797F5623F8442C2CE...",
      "key": "6559CA840CF8360A8AF7...",
      "ct": "0A17C2F7A82BBDE588262..."
    }]
  }, {
    "tgId": 3,
    "direction": "encrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 61,
      "iv": "4AAF5D6F6E25B8A868D8D...",
      "key": "0D0949FB32A2DC6BA267...",
      "pt": "98EE9313512D5BEC19715..."
    }]
  }, {
    "tgId": 4,
    "direction": "decrypt",
    "testType": "MCT",
    "keyLen": 128,
    "tests": [{
      "tcId": 64,
      "iv": "663D4E1B6F09FE1935E69...",
      "key": "5924D41588E2DC657514...",
      "ct": "83C1C3AF23A3F658DF142..."
    }]
  }]
}]
```

Figure C-21

The following shows AES-CFB128 AFT and MCT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2059,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "1C9BF58FF640041F8E..."
        },
        {
          "tcId": 2,
          "ct": "2C822934B8D747336..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "4BC37D318900379CD75..."
        },
        {
          "tcId": 32,
          "pt": "523057EC2E120826..."
        }
      ]
    },
    {
      "tgId": 3,
      "tests": [{
        "tcId": 61,
        "resultsArray": [{
            "key": "0D0949FB32A...",
            "iv": "4AAF5D6F6E25...",
            "pt": "98EE9313512D...",
            "ct": "7E94144C4DD4..."
          },
          {
            "key": "739D5DB77F7...",
            "iv": "7E94144C4DD4...",
            "pt": "E93E4CCB2BD1...",
            "ct": "050CE71D2451..."
          }
        ]
      }]
    },
    {
      "tgId": 4,
      "tests": [{
        "tcId": 64,
        "resultsArray": [{
            "key": "5924D41588E...",
            "iv": "663D4E1B6F09...",
            "ct": "83C1C3AF23A3...",
            "pt": "32D4D152D488..."
          },
          {
            "key": "6BF005475C6...",
            "iv": "32D4D152D488...",
            "ct": "3CC4191B8EBE...",
            "pt": "BB97ADEF9F08..."
          }
        ]
      }]
    }
  ]
}]
```

Figure C-22

The following shows AES-CTR AFT and counter request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2066,
  "algorithm": "ACVP-AES-CTR",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "direction": "encrypt",
    "keyLen": 128,
    "testType": "AFT",
    "tests": [{
      "tcId": 1,
      "key": "E870131CE703D6514E761F95E6EE9EFB",
      "payloadLen": 128,
      "iv": "53F225D8DE97F14BFE3EC65EC3FFF7D3",
      "pt": "91074131F1F86CCD548D22A69340FF39"
    }, {
      "tcId": 2,
      "key": "2C759788A49BF060353344413A1D0FFC",
      "payloadLen": 128,
      "iv": "A4DE6D846C3AE5D5FF78163FF209AFE4",
      "pt": "BA37A61FD041F2881921D4705AD329DD"
    }]
  }, {
    "tgId": 2,
    "direction": "decrypt",
    "keyLen": 128,
    "testType": "AFT",
    "tests": [{
      "tcId": 31,
      "key": "51B4375D6FB348A55477E3C3163F59C7",
      "payloadLen": 128,
      "iv": "93893A056C6C6F866A04D657A544F1F8",
      "ct": "F2FF4B0C2E771A41525EA67AD036B459"
    }, {
      "tcId": 32,
      "key": "6A4F0B775490D554F19B5A061A362666",
      "payloadLen": 128,
        "iv": "9877D2AB7568CEF28BA945B046BA20BE",
        "ct": "09F4EEF2322BE13D75FF6DA86E8617B5"
      }]
    }, {
      "tgId": 3,
      "direction": "encrypt",
      "keyLen": 128,
      "testType": "CTR",
      "tests": [{
        "tcId": 829,
        "pt": "CE8E4B6F7C68DE5FDE3...",
        "iv": "00000000000000000000000000000039",
        "key": "3A9A8485E1B7BA1987F88F8C095257C4"
      }]
    }]
}]
```

Figure C-23

The following shows AES-CTR AFT and counter responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2066,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "3AF64C7037EE4813D8..."
        },
        {
          "tcId": 2,
          "ct": "2DFDFCDDC4CFD3CBCE..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 31,
          "pt": "349012E0807CA95CA5..."
        },
        {
          "tcId": 32,
          "pt": "2986D4B3FB208F0189..."
        }
      ]
    },
    {
      "tgId": 3,
      "tests": [{
        "tcId": 829,
        "ct": "676EC652D5B095136..."
      }]
    }
  ]
}]
```

Figure C-24

The following shows AES-CTR RFC3686 request vectors with internal iv generation.

```
{
 "vsId": 1,
 "algorithm": "ACVP-AES-CTR",
 "revision": "1.0",
 "testGroups": [{
   "tgId": 1,
   "testType": "AFT",
   "direction": "encrypt",
   "keyLen": 128,
   "tests": [{
     "tcId": 12,
     "pt": "3687D763A3EEC3E3099678068F3CDEB4C7B12BA83C50CCB744D8945C0DB0078E",
     "payloadLen": 256,
     "key": "208A474D7567BF87A1A62D0767724547"
    },
    {
     "tcId": 13,
     "pt":
 "F96CBC81F0B876A463FD467C5FBA19791A1BE394DF61C883BE7ECB67270846A1E345991F81DAAE4532DEFCFC
     "payloadLen": 384,
     "key": "629EFE9A344A081CD4E4D758C1E759BF"
    }
   ]
  },
  {
   "tgId": 3,
   "testType": "AFT",
   "direction": "decrypt",
   "keyLen": 128,
   "tests": [{
     "tcId": 62,
     "payloadLen": 256,
     "ct": "0306E3B1F1719CD7C64296F52B06F246CAA463BE19309AC2CF842ADE0B0BCD21",
     "iv": "14FD559C120735498CE09BA800000001",
     "key": "4D94B2155A6322DB76878C71763EE544"
    },
    {
     "tcId": 63,
     "payloadLen": 384,
     "ct":
 "474AB381112D9FB530BC2E0B2B7E6D139243BC6B1D23D21508E18E82D85218DB10C3C4DABE278B2D334860BE
     "iv": "8E0759C1B24B8CCBDE7D51C100000001",
     "key": "BE4AD76083C6A471803FFE704B6F194D"
    }
   ]
  }
 ]
}
```

Figure C-25

The following shows AES-CTR RFC3686 with internal iv generation responses.

```
{
 "vsId": 0,
 "algorithm": "ACVP-AES-CTR",
 "mode": "",
 "revision": "1.0",
 "isSample": true,
 "testGroups": [{
   "tgId": 1,
   "tests": [{
     "tcId": 12,
     "ct": "00F5E67B3C8C6038D907D5866866DDBE583CF90DD8AE37159D1CC0235EEA6175",
     "iv": "992AA770156A7E1AB58BAFCA00000001"
    },
    {
     "tcId": 13,
     "ct":
 "7EA7DC8A993C1EBA61239CFBBBA4244DB185C4F8F248CCF1F1AF7DD7993B1AE5EE05D51AC58D453FE32EC596
     "iv": "F083A4495B66E5DF0607BF9200000001"
    }
   ]
  },
  {
   "tgId": 3,
   "tests": [{
     "tcId": 62,
     "pt": "0FC0CAF36921D2803DC9AEBFD5473124D77969BA9FF813861332A3E77E9265C1"
    },
    {
     "tcId": 63,
     "pt":
 "9C0DA9C01AEB0B4A42E99776A4E2DFE8E50A7A602E257C7D2EB3C79A529BAA4AC130ADBE6537CB39AA068500
    }
   ]
  }
 ]
}
```

Figure C-26

The following shows AES-CTR RFC3686 request vectors with external iv generation.

```
{
 "vsId": 1,
 "algorithm": "ACVP-AES-CTR",
 "revision": "1.0",
 "testGroups": [{
   "tgId": 1,
   "testType": "AFT",
   "direction": "encrypt",
   "keyLen": 128,
   "tests": [{
     "tcId": 12,
     "pt": "3687D763A3EEC3E3099678068F3CDEB4C7B12BA83C50CCB744D8945C0DB0078E",
          "iv": "992AA770156A7E1AB58BAFCA00000001",
     "payloadLen": 256,
     "key": "208A474D7567BF87A1A62D0767724547"
    },
    {
     "tcId": 13,
     "pt":
 "F96CBC81F0B876A463FD467C5FBA19791A1BE394DF61C883BE7ECB67270846A1E345991F81DAAE4532DEFCFC
          "iv": "F083A4495B66E5DF0607BF9200000001",
     "payloadLen": 384,
     "key": "629EFE9A344A081CD4E4D758C1E759BF"
    }
   ]
  },
  {
   "tgId": 3,
   "testType": "AFT",
   "direction": "decrypt",
   "keyLen": 128,
   "tests": [{
     "tcId": 62,
     "payloadLen": 256,
     "ct": "0306E3B1F1719CD7C64296F52B06F246CAA463BE19309AC2CF842ADE0B0BCD21",
     "iv": "14FD559C120735498CE09BA800000001",
     "key": "4D94B2155A6322DB76878C71763EE544"
    },
    {
     "tcId": 63,
     "payloadLen": 384,
     "ct":
 "474AB381112D9FB530BC2E0B2B7E6D139243BC6B1D23D21508E18E82D85218DB10C3C4DABE278B2D334860BE
     "iv": "8E0759C1B24B8CCBDE7D51C100000001",
     "key": "BE4AD76083C6A471803FFE704B6F194D"
    }
   ]
  }
 ]
}
```

Figure C-27

The following shows AES-CTR RFC3686 with external iv generation responses.

```
{
 "vsId": 0,
 "algorithm": "ACVP-AES-CTR",
 "mode": "",
 "revision": "1.0",
 "isSample": true,
 "testGroups": [{
   "tgId": 1,
   "tests": [{
     "tcId": 12,
     "ct": "00F5E67B3C8C6038D907D5866866DDBE583CF90DD8AE37159D1CC0235EEA6175"
    },
    {
     "tcId": 13,
     "ct":
 "7EA7DC8A993C1EBA61239CFBBBA4244DB185C4F8F248CCF1F1AF7DD7993B1AE5EE05D51AC58D453FE32EC596
    }
   ]
  },
  {
   "tgId": 3,
   "tests": [{
     "tcId": 62,
     "pt": "0FC0CAF36921D2803DC9AEBFD5473124D77969BA9FF813861332A3E77E9265C1"
    },
    {
     "tcId": 63,
     "pt":
 "9C0DA9C01AEB0B4A42E99776A4E2DFE8E50A7A602E257C7D2EB3C79A529BAA4AC130ADBE6537CB39AA068500
    }
   ]
  }
 ]
}
```

Figure C-28

The following shows AES-XPN AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "algorithm": "ACVP-AES-XPN",
  "revision": "1.0",
  "vsId": 1,
  "testGroups": [
    {
      "tgId": 1,
      "testType": "AFT",
      "direction": "encrypt",
      "keyLen": 128,
      "ivLen": 96,
      "ivGen": "external",
      "ivGenMode": "8.2.2",
      "saltLen": 96,
      "saltGen": "external",
      "payloadLen": 128,
      "aadLen": 120,
      "tagLen": 64,
      "tests": [
        {
          "tcId": 1,
          "plainText": "4849547C706231E248148...",
          "key": "4A23FDD31C1B321C1D3E1A74ECA9585A",
          "aad": "6B55B1B784180DE574F7709E480273",
          "iv": "A05134709620EAB47DE77FCB",
          "salt": "F0C77CB78D20BBDCF3A3C5EB"
        },
        {
          "tcId": 2,
          "plainText": "BF1D8173DA7F0273B7DA8...",
          "key": "254E5AFE555D807E5ECC2FFAB2E3E107",
          "aad": "304A2EC82959B419B8852F5C6A09D1",
          "iv": "1BA39F6A71F075FEB72B91D6",
          "salt": "AF44CD3E80088B8FD252AAB0"
        }
      ]
    }
  ]
}]
```

Figure C-29

The following shows AES-XPN AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 1,
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "testPassed": false
        },
        {
          "tcId": 2,
          "cipherText": "D3104958599BE7BB9E672F...",
          "tag": "48408062AA84718B"
        }
      ]
    }
  ]
}]
```

Figure C-30

The following shows AES-XTS 1.0 AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2065,
  "algorithm": "ACVP-AES-XTS",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "testType": "AFT",
    "direction": "encrypt",
    "keyLen": 128,
    "tweakMode": "hex",
    "payloadLen": 65536,
    "tests": [{
      "tcId": 1,
      "key": "2866E3659E11C7890313EDAC9...",
      "tweakValue": "C7850E1C99DA28C5E7...",
      "pt": "03F912D53EA625A7D206002864..."
    }, {
      "tcId": 2,
      "key": "98B66C26FF9E4EF2BCBC3A212...",
      "tweakValue": "57B127C8DAD60138C5...",
      "pt": "20D7E083519F39DB185CDA2397..."
    }]
  }, {
    "tgId": 2,
    "testType": "AFT",
    "direction": "decrypt",
    "keyLen": 128,
    "tweakMode": "hex",
    "payloadLen": 65536,
    "tests": [{
      "tcId": 101,
      "key": "BB626CADBBFB907AC5C795080...",
      "tweakValue": "8B7E45A9200BDC72EB...",
      "ct": "B85B91029478C3E02EBC619EC7..."
    }, {
      "tcId": 102,
      "key": "9B859C56C1542C19F29AA7A4F...",
      "tweakValue": "99FE35549768F476E2...",
      "ct": "53CEE8379B03A38E33CCCC6EA0..."
    }]
  }]
}]
```

Figure C-31

The following shows AES-XTS 1.0 AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2065,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "97ED8057287E4FD0E1..."
        },
        {
          "tcId": 2,
          "ct": "BCACA25E6A625DB16..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 101,
          "pt": "8AD40CBE09CD92FB0..."
        },
        {
          "tcId": 102,
          "pt": "07DD39402F4D427D7..."
        }
      ]
    }
  ]
}]
```

Figure C-32

The following shows AES-XTS 2.0 AFT request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2065,
  "algorithm": "ACVP-AES-XTS",
    "revision": "2.0",
  "testGroups": [{
    "tgId": 1,
    "testType": "AFT",
    "direction": "encrypt",
    "keyLen": 128,
    "tweakMode": "hex",
    "tests": [{
      "tcId": 1,
      "key": "2866E3659E11C7890313EDAC9...",
      "tweakValue": "C7850E1C99DA28C5E7...",
      "pt": "03F912D53EA625A7D206002864...",
        "payloadLen": 2048,
        "dataUnitLen": 1024
      }, {
        "tcId": 2,
        "key": "98B66C26FF9E4EF2BCBC3A212...",
        "tweakValue": "57B127C8DAD60138C5...",
        "pt": "20D7E083519F39DB185CDA2397...",
        "payloadLen": 2048,
        "dataUnitLen": 1024
      }]
    }, {
      "tgId": 2,
      "testType": "AFT",
      "direction": "decrypt",
      "keyLen": 128,
      "tweakMode": "hex",
      "tests": [{
        "tcId": 101,
        "key": "BB626CADBBFB907AC5C795080...",
        "tweakValue": "8B7E45A9200BDC72EB...",
        "ct": "B85B91029478C3E02EBC619EC7...",
        "payloadLen": 2048,
        "dataUnitLen": 1024
      }, {
        "tcId": 102,
        "key": "9B859C56C1542C19F29AA7A4F...",
        "tweakValue": "99FE35549768F476E2...",
        "ct": "53CEE8379B03A38E33CCCC6EA0...",
        "payloadLen": 2048,
        "dataUnitLen": 1024
      }]
    }]
}]
```

Figure C-33

The following shows AES-XTS 2.0 AFT responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2065,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "97ED8057287E4FD0E1..."
```

```
        },
        {
          "tcId": 2,
          "ct": "BCACA25E6A625DB16..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 101,
          "pt": "8AD40CBE09CD92FB0..."
        },
        {
          "tcId": 102,
          "pt": "07DD39402F4D427D7..."
        }
      ]
    }
  ]
}]
```

Figure C-34

The following shows AES-KW request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2063,
  "algorithm": "ACVP-AES-KW",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "testType": "AFT",
    "direction": "encrypt",
    "kwCipher": "cipher",
    "keyLen": 128,
    "payloadLen": 192,
    "tests": [{
      "tcId": 1,
      "key": "71389B09A3EA1AAE1F265CD3DE8FABB7",
      "pt": "3D90BE277A057C024A485F02486D733..."
    }, {
      "tcId": 2,
      "key": "B75DB6D92A66A3E8E991FEDBA3DAACA7",
      "pt": "3323EC2514C2902C424ABE968CA09FD..."
    }]
  }, {
    "tgId": 2,
    "testType": "AFT",
    "direction": "decrypt",
    "kwCipher": "cipher",
    "keyLen": 128,
    "payloadLen": 192,
    "tests": [{
      "tcId": 901,
      "key": "E5319E0061F89DE08CB590EA...",
      "ct": "1DE720863C759EC0682429AA4..."
    }, {
      "tcId": 902,
      "key": "D16C5C5FDE26C1962342AACF...",
      "ct": "F2EC43D61F2F356E1B2850D7C..."
    }]
  }]
}]
```

Figure C-35

The following shows AES-KW responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2063,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "BD009027DA8F4176B..."
        },
        {
          "tcId": 2,
          "ct": "B8BB3D3C76FDFD359..."
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 901,
          "pt": "A6BA646D0D33808AB..."
        },
        {
          "tcId": 902,
          "pt": "B40AC5F6ED5A706CB..."
        }
      ]
    }
  ]
}]
```

Figure C-36

The following shows AES-KWP request vectors.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2064,
  "algorithm": "ACVP-AES-KWP",
    "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "testType": "AFT",
    "direction": "encrypt",
    "kwCipher": "cipher",
    "keyLen": 128,
    "payloadLen": 808,
    "tests": [{
      "tcId": 1,
      "key": "EE3B424525EE1B2D0B8CDC4CCB15F018",
      "pt": "269701A6DE9A2E8A8B2E28027..."
    }, {
      "tcId": 2,
      "key": "579C5EBBD1D07F828251FE567326C5DD",
      "pt": "634945E0FD1FA2E733CD60462..."
    }]
  }, {
    "tgId": 2,
    "testType": "AFT",
    "direction": "decrypt",
    "kwCipher": "cipher",
    "keyLen": 128,
    "payloadLen": 808,
    "tests": [{
      "tcId": 301,
      "key": "0EB557E0F938E08662EB9EDAAE05725F",
      "ct": "1BB87C360F2B644CD0BC75369..."
    }, {
      "tcId": 302,
      "key": "644E2869C9698ADBB4417A8ED65748DC",
      "ct": "583741B7624759F37EED76F76..."
    }]
  }]
}]
```

Figure C-37

The following shows AES-KWP responses.

```
[{
  "acvVersion": "{acvp-version}"
},{
  "vsId": 2064,
  "testGroups": [{
        "tgId": 1,
        "tests": [{
            "tcId": 1,
            "ct": "58385237F04FD67F0..."
          },
          {
            "tcId": 2,
            "ct": "0D6FE2D0A8605981E..."
          }
        ]
      },
      {
        "tgId": 2,
        "tests": [{
            "tcId": 300,
            "ct": "D2A239230130B6077..."
          },
          {
            "tcId": 301,
            "testPassed": false
          }
        ]
      }
    ]
}]
```

Figure C-38

The following shows AES-FF1 request vectors.

```
{
  "vsId": 42,
  "algorithm": "ACVP-AES-FF1",
  "revision": "1.0",
  "isSample": false,
  "testGroups": [{
      "tgId": 1,
      "testType": "AFT",
      "direction": "encrypt",
      "keyLen": 128,
      "alphabet": "0123456789",
      "radix": 10,
      "tests": [{
          "tcId": 1,
          "tweak": "",
          "tweakLen": 0,
          "pt": "5989891000",
          "key": "FA407521178EDB931997C9EF5FF4F8BB"
        },
        {
          "tcId": 2,
          "tweak": "CB81CF732B22A983B2C6E584726C9F59",
          "tweakLen": 128,
          "pt": "6045438460218079668054470789645161855775615270273458716 1",
          "key": "E3EFDAF1ABEA7863A0A95F833420D083"
        }
      ]
    },
    {
      "tgId": 2,
      "testType": "AFT",
      "direction": "encrypt",
      "keyLen": 128,
      "alphabet": "abcdefghijklmnopqrstuvwxyz",
      "radix": 26,
      "tests": [{
          "tcId": 26,
          "tweak": "",
          "tweakLen": 0,
          "pt": "zlwagydvpt",
          "key": "D263686051802ECAE0217F4123000376"
        },
        {
          "tcId": 27,
          "tweak": "994C168B9F6225C4BC12A83561C0E1A6",
          "tweakLen": 128,
          "pt": "uxmdsdjbsywthsvzjlfcwlmpkarnaeoirtihgfuu",
          "key": "53CA14AF6F97612C96FFAA2BA8D88C44"
        }
      ]
    },
    {
      "tgId": 10,
      "testType": "AFT",
      "direction": "decrypt",
      "keyLen": 128,
      "alphabet": "0123456789",
      "radix": 10,
      "tests": [{
          "tcId": 226,
          "tweak": "",
          "tweakLen": 0,
          "key": "82DD08D210EB34C9743EC102E058CEE3",
          "ct": "8416752187"
        },
        {
          "tcId": 227,
          "tweak": "03130ABD79425EEC806617434C60B2FE",
          "tweakLen": 128,
          "key": "E6265A6503AD2F4F13FCCF8B8AD64638",
          "ct": "26379838628949309091263132236041132252667968597137110245"
        }
      ]
    }
  ]
}
```

Figure C-39

The following shows AES-FF1 responses.

```
{
  "vsId": 42,
  "algorithm": "ACVP-AES-FF1",
  "revision": "1.0",
  "isSample": false,
  "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 1,
          "ct": "4896500946"
        },
        {
          "tcId": 2,
          "ct": "69747385701019112488208222409900597881359856066248208863"
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [{
          "tcId": 26,
          "ct": "odmrhltvlj"
        },
        {
          "tcId": 27,
          "ct": "lifbvigwtcwmkiucogaztntcagaqqtoioagwsgef"
        }
      ]
    },
    {
      "tgId": 18,
      "tests": [{
          "tcId": 426,
          "pt": "/HN6wiTZoc"
        },
        {
          "tcId": 427,
          "pt": "NyFsYHaUg0000JcSKJhRvOe0000E"
        }
      ]
    }
  ]
}
```

Figure C-40

The following shows AES-FF3-1 request vectors.

```
{
  "vsId": 42,
  "algorithm": "ACVP-AES-FF3-1",
  "revision": "1.0",
  "isSample": false,
  "testGroups": [
    {
      "tgId": 1,
      "testType": "AFT",
      "direction": "encrypt",
      "keyLen": 128,
      "alphabet": "0123456789",
      "radix": 10,
      "tests": [
        {
          "tcId": 1,
          "tweak": "CBD09280979564",
          "tweakLen": 56,
          "pt": "3992520240",
          "key": "2DE79D232DF5585D68CE47882AE256D6"
        },
        {
          "tcId": 2,
          "tweak": "C4E822DCD09F27",
          "tweakLen": 56,
          "pt": "60761757463116869318437658042297305934914824457484538562",
          "key": "01C63017111438F7FC8E24EB16C71AB5"
        }
    ]
  }
   ]
}
```

Figure C-41

The following shows AES-FF-31 responses.

```
{
  "vsId": 42,
  "algorithm": "ACVP-AES-FF3-1",
  "revision": "1.0",
  "isSample": false,
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "ct": "8901801106"
        },
        {
          "tcId": 2,
          "ct": "35637144092473838892796702739628394376915177448290847293"
        }
    ]
  }
   ]
}
```

Figure C-42

Appendix D — Example TDES Test and Results JSON Object

The following is a example JSON object for test vectors sent from the ACVP server to the crypto module for an TDES-ECB algorithm functional test.

```
[{
  "acvVersion": "{acvp-version}"
},{
    "vsId": 1564,
    "algorithm": "ACVP-TDES-ECB",
    "revision": "1.0",
    "testGroups": [{
        "tgId": 1,
        "direction": "encrypt",
        "testType": "AFT",
        "tests": [{
            "tcId": 236,
            "key1": "5BE5B5FE9BB3E36D",
            "key2": "26E92C6DD35D7AB3",
            "key3": "4F89ADAD15D62FE3",
            "pt": "7119CCA0648787AE"
        }, {
            "tcId": 237,
            "key1": "2C7015EC2C044591",
            "key2": "230D79A1D0F2469D",
            "key3": "7A9EF7FDC4383131",
            "pt": "772923F53BA2EA60E7AE232..."
        }]
    }]
}]
```

Figure D-1

The following is a example JSON object for test results sent from the crypto module to the ACVP server for an TDES-ECB algorithm functional test.

```
[{
  "acvVersion": "{acvp-version}"
},{
    "vsId": 1564,
    "testGroups": [{
        "tgId": 1,
        "tests": [{
                "tcId": 236,
                "ct": "1E85F8256575B8B1"
            },
            {
                "tcId": 237,
                "ct": "BEFD0E02088D48648FEBAAF..."
            }
        ]
    }]
}]
```

Figure D-2

The following is a example JSON object for test vectors sent from the ACVP server to the crypto module for the TDES-CFB1 algorithm functional test.

```
[{
  "acvVersion": "{acvp-version}"
},{
    "vsId": 1564,
    "algorithm": "ACVP-TDES-CFB1",
    "revision": "1.0",
    "testGroups": [{
            "tgId": 1,
            "direction": "encrypt",
            "testType": "AFT",
            "keyingOption": 1,
            "tests": [{
                "tcId": 1,
                "key1": "1046913489980131",
                "key2": "1046913489980131",
                "key3": "1046913489980131",
                "pt": "00",
                "payloadLen": 1,
                "iv": "0000000000000000"
            }]
        },
        {
            "tgId": 2,
            "direction": "encrypt",
            "testType": "MCT",
            "keyingOption": 1,
            "tests": [{
                "tcId": 961,
                "key1": "337C857E01DE54B7",
                "key2": "F106296828FCCA0D",
                "key3": "2F65BF5A655FFFA3",
                "pt": "80",
                "payloadLen": 1,
                "iv": "0C4CCC40D9C8C5D7"
            }]
        }
    ]
}]
```

Figure D-3

The following is a example JSON object for test results sent from the crypto module to the ACVP server for an TDES-CFB1 algorithm functional test.

```
[{
  "acvVersion": "{acvp-version}"
},{
    "vsId": 1564,
    "testGroups": [{
            "tgId": 1,
            "tests": [{
                "tcId": 1,
                "ct": "00"
            }]
        },
        {
            "tgId": 2,
            "tests": [{
                "tcId": 961
                "resultsArray": [{
                        "key1": "337C857E01DE54B7",
                        "key2": "F106296828FCCA0D",
                        "key3": "2F65BF5A655FFFA3",
                        "pt": "80",
                        "ct": "00",
                        "iv": "0C4CCC40D9C8C5D7"
                    },
                    {
                        "key1": "290E7326C8833420",
                        "key2": "8FE6BF67EF0B2325",
                        "key3": "3E2976E05EB0646D",
                        "pt": "80",
                        "ct": "80",
                        "iv": "1A73F758C95C6196"
                    }
                ]
            }]
        }
    ]
}]
```

Figure D-4

Appendix E — Example TDES MCT Test and Results JSON Object

The following is a example JSON object for test vectors sent from the ACVP server to the crypto module for an TDES-ECB Monte Carlo test.

```
[{
  "acvVersion": "{acvp-version}"
},{
    "vsId": 1564,
    "algorithm": "ACVP-TDES-ECB",
    "revision": "1.0",
    "testGroups": [{
        "tgId": 1,
        "direction": "encrypt",
        "testType": "MCT",
        "tests": [{
            "tcId": 492,
            "key1": "0EABB0E6B0F129D5",
            "key2": "DF61EAD07315DA37",
            "key3": "EFA2B6A252A18694",
            "ct": "2970B363C1461FAF"
        }]
    }]
}]
```

Figure E-1

The following is a example JSON object for test results sent from the crypto module to the ACVP server for an TDES-ECB Monte Carlo test, only 2 iterations shown for brevity. For MCT results of each iteration are fed into the next iteration. Therefore the results carry all fields to assist in any failure diagnosis.

```
[{
  "acvVersion": "{acvp-version}"
},{
    "vsId": 1564,
    "testGroups": [{
        "tgId": 1,
        "tests": [{
            "tcId": 492,
            "resultsArray": [{
                    "key1": "0EABB0E6B0F129D5",
                    "key2": "DF61EAD07315DA37",
                    "key3": "EFA2B6A252A18694",
                    "ct": "2970B363C1461FAF",
                    "pt": "40F806F9DE3466C0"
                },
                {
                    "key1": "4F52B61F6EC4...",
                    "key2": "2FEC373726FE...",
                    "key3": "37B57029B65B...",
                    "ct": "40F806F9DE3466C0",
                    "pt": "A498B9748F2FB1E5"
                }
            ]
        }]
    }]
}]
```

Figure E-2