ACVP Message Authentication Algorithm JSON Specification

Barry Fussell
Cisco Systems, Inc.
170 West Tasman Drive, San Jose, California

Russell Hammett
G2, Inc.
302 Sentinel Drive, Suite #300, Annapolis Junction, MD 20701

August 01, 2018

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

Abstract

This document defines the JSON schema for testing HMAC, CMAC, and GMAC implementations with the ACVP specification.

Keywords

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

ACVP; cryptography

Foreword

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Audience

This document is intended for the users and developers of ACVP.

Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 of [RFC 2119] and [RFC 8174] when, and only when, they appear in all capitals, as shown here.

Acknowledgements

This document is produced by the Security Testing, Validation and Measurement group under the Automated Cryptographic Validation Testing (ACVT) program.

Executive Summary

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 HMAC, CMAC, and GMAC implementations using ACVP.

Disclaimer

Additional Information

For additional information on NIST's Cybersecurity programs, projects and publications, visit the Computer Security Resource Center. Information on other efforts at NIST and in the Information Technology Laboratory (ITL) is also available.

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 HMAC, CMAC, and GMAC implementations using ACVP.

2. Supported HMAC, CMAC, and GMAC Algorithms

The following Message Authentication Code Algorithms **MAY** be advertised by the ACVP compliant cryptographic module:

- HMAC-SHA-1
- HMAC-SHA2-224
- HMAC-SHA2-256
- HMAC-SHA2-384
- HMAC-SHA2-512
- HMAC-SHA2-512/224
- HMAC-SHA2-512/256
- HMAC-SHA3-224
- HMAC-SHA3-256
- HMAC-SHA3-384
- HMAC-SHA3-512
- CMAC-AES
- CMAC-TDES
- ACVP-AES-GMAC

3. Test Types and Test Coverage

The ACVP server performs a set of tests on the MAC algorithms 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 CMAC-AES, CMAC-TDES, HMAC-SHA-1, HMAC-SHA2-256, etc. This section describes the design of the tests used to validate implementations of MAC algorithms.

3.1. Test Types

There is a single test type for MACs (broken into subsections for CMACs). the single test type, algorithm functional test (AFT) can be described as follows:

- "AFT"—Algorithm Function Test. The IUT processes all of HMAC, GMAC and the "gen" direction of CMAC by running the randomly chosen key and message data (with constraints as per the IUT's capabilities registration) through the MAC algorithm. CMAC has an additional "ver" direction present in its testing to ensure the IUT can successfully determine when a MAC does not match its originating message/key combination.

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [SP 800-38B] and [FIPS 198-1].

3.2.1. CMAC Requirements Covered

- SP 800-38B Section 6.2 Mac Generation. ACVP server creates random sets of keys and messages for the IUT to process, then compares the IUT's result to the ACVP result.
- SP 800-38B Section 6.3 Mac Verification. ACVP server creates random sets of keys, messages, and macs. These test vectors are then randomly altered to ensure the MAC will not match the given key and message. Using the provided test case, the IUT is expected to validate the mac against the provided key and message.

3.2.2. CMAC Requirements Not Covered

- SP 800-38B Section 5.4 Sub-keys. While sub-keys are computed, as they are intermediate values, are not validated via current testing.
- SP 800-38B Section 5.5 Input and Output Data. The 'mlen' is provided and not inferred.
- SP 800-38B Appendix A. Length of MAC. The ACVP server will generate vectors as per the IUT's specified criteria. The IUT **SHOULD** register its entire range of supported MAC lengths, regardless of security strength. The ACVP server will test a random sampling of valid MAC lengths as per the IUT registration—this generally includes the minimum and maximum MAC length.

3.2.3. GMAC Requirements Covered

- SP 800-38D Section 5.1 Block Cipher. ACVP testing **SHALL** make use of the AES block cipher when testing GMAC.

- SP 800-38D Section 5.2 Two Gcm Functions. ACVP testing **MAY** test both the generate and verify functions of GCM (without making use of a payload) to help ensure a proper implementation. The ACVP and IUT **MAY** test the encrypt (generate) and decrypt (verify) utilizing a key, IV/nonce, and AAD as described in this document section.
- SP 800-38D Section 6 Mathematical Components of GCM. GHASH and GCTR produce intermediate values and **SHALL** be (indirectly) evaluated for correctness via the ACVP generated GMAC test vectors.
- SP 800-38D Section 7 GCM Specification. When the IUT registers a direction capability of “encrypt”, the ACVP server **MUST** generate vectors for the GCM-AE function. When the IUT registers a capability of “decrypt”, the ACVP server **MUST** generate test vectors for the GCM-AD function.

3.2.4. GMAC Requirements Not Covered

- SP 800-38D Section 5.2.1.1 Input Data. GMAC **MUST NOT** make use of a plaintext.
- SP 800-38D Section 5.2.1.2 Output Data. GMAC **MUST NOT** make use of a ciphertext.
- SP 800-38D Section 7 GCM Specification. The ACVP server **MUST NOT** make use of a plaintext or ciphertext for the generation of test vectors for use in GMAC testing.
- SP 800-38D Section 8 Uniqueness Requirement on IVs and Keys. Key establishment, IV construction, or number of invocations for a specific key/IV **SHALL NOT** be tested under the scope of the ACVP testing.

3.2.5. HMAC Requirements Covered

- FIPS 198-1 Section 3 Cryptographic Keys. The ACVP server will test, depending on the nature of the IUT capabilities registration, keys that are below, at, or above the hashing algorithm block size.
- FIPS 198-1 Section 4 HMAC Specification. Mac Generation. ACVP server creates random sets of keys and messages for the IUT to process, then compares the IUT’s result to the ACVP result.
- FIPS 198-1 Section 5 Truncation. The ACVP server is capable of generating MACs as per the capability registration of the IUT. Groups will be created containing a random sampling of valid MAC lengths from the IUT registration.

3.2.6. HMAC Requirements Not Covered

N/A

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 MAC 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 1

4.2. Required Prerequisite Algorithms for MAC Validations

Each MAC implementation relies on other cryptographic primitives. For example, HMAC uses an underlying SHA 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 MAC Prerequisite Algorithms JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	a prerequisite algorithm	string	AES, SHA, TDES with associated mode such as ECB, GCM or digest size
valValue	algorithm validation number	string	Actual number or "same"
prereqAlgVal	prerequisite algorithm validation	object with algorithm and valValue properties	See above

4.3. MAC Algorithm Registration Properties

Each MAC algorithm capability advertised is a self-contained JSON object using the following values.

Table 3 — MAC Algorithm Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	The MAC algorithm to be validated	string	See Section 2
revision	The algorithm testing revision to use	string	"1.0"
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See Section 4.2
capabilities	The individual MAC capabilities	array of capability objects	See Section 5.1.1, Section 6.1.1, Section 7.1, or Section 8.1

5. CMAC-AES

5.1. CMAC-AES Algorithm Capabilities Registration

Each CMAC-AES algorithm capability advertised is a self-contained JSON object using Table 3. Each capability object describes a separate permutation of direction and key length.

5.1.1. CMAC-AES Capability Details

Table 4 — CMAC-AES Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
direction	The MAC direction(s) to test	array	"gen" and/or "ver"
keyLen	The keyLen supported	array of integer	[128, 192, 256]
msgLen	The CMAC message lengths supported in bits, values **MUST** be mod 8	Domain	Min: 0, Max: 524288, Inc: 8
macLen	The supported mac sizes	Domain	Min: 1, Max: 128

'macLen' for CMAC is a Domain of values, the server **MAY** choose values defined by these rules:

- The smallest CMAC length supported
- A second CMAC length supported
- The largest CMAC length supported

'msgLen' for CMAC is a Domain of values, the server **MAY** choose values defined by these rules:

- The smallest message length supported
- Two message lengths divisible by block size
- Two message lengths NOT divisible by block size
- The largest message length supported

5.1.2. Example CMAC-AES Capabilities JSON Object

The following is an example JSON object advertising support for CMAC-AES.

```
{
  "algorithm": "CMAC-AES",
  "revision": "1.0",
  "capabilities": [
    {
      "direction": ["gen", "ver"],
      "keyLen": [128],
      "msgLen": [
```

```
          {
            "min": 0,
            "max": 65536,
            "increment": 8
          }
        ],
        "macLen": [
          {
            "min": 64,
            "max": 128,
            "increment": 8
          }
        ]
      },
      {
        "direction": ["gen", "ver"],
        "keyLen": [192],
        "msgLen": [
          {
            "min": 0,
            "max": 65536,
            "increment": 8
          }
        ],
        "macLen": [
          {
            "min": 64,
            "max": 128,
            "increment": 8
          }
        ]
      },
      {
        "direction": ["gen", "ver"],
        "keyLen": [256],
        "msgLen": [
          {
            "min": 0,
            "max": 65536,
            "increment": 8
          }
        ],
        "macLen": [
          {
            "min": 64,
            "max": 128,
```

```
          "increment": 8
        }
      ]
    }
  ]
}
```


Figure 2

6. CMAC-TDES

6.1. CMAC-TDES Algorithm Capabilities Registration

Each CMAC-TDES algorithm capability advertised is a self-contained JSON object using Table 3. Each capability object describes a separate permutation of direction and key length.

6.1.1. CMAC-TDES Capability Details

Table 5 — CMAC-TDES Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
direction	The MAC direction(s) to test	array	"gen", "ver"
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 only suitable for decrypt (K1, K2, K1). Keying option 3 (No longer valid for testing, save TDES KATs) is a single key, now deprecated (K1, K1, K1).	array	1, 2
msgLen	The CMAC message lengths supported in bits, values **MUST** be mod 8	Domain	Min: 0, Max: 524288, Inc: 8
macLen	The supported mac sizes	Domain	Min: 32, Max: 64

'macLen' for CMAC contains a Domain of values, the server may choose values defined by these rules:

- The smallest CMAC length supported
- A second CMAC length supported
- The largest CMAC length supported

'msgLen' for CMAC contains a Domain of values, the server may choose values defined by these rules:

- The smallest message length supported
- Two message lengths divisible by block size
- Two message lengths NOT divisible by block size
- The largest message length supported

6.1.2. Example CMAC-TDES Capabilities JSON Object

The following is an example JSON object advertising support for CMAC-TDES.

```
{
  "algorithm": "CMAC-TDES",
  "revision": "1.0",
  "capabilities": [
    {
      "direction": ["gen", "ver"],
      "keyingOption": [1],
      "msgLen": [
        {
          "min": 0,
          "max": 65536,
          "increment": 8
        }
      ],
      "macLen": [
        {
          "min": 32,
          "max": 64,
          "increment": 8
        }
      ]
    }
  ]
}
```


Figure 3

7. GMAC

7.1. AES-GMAC Algorithm Capabilities Registration

Each algorithm capability advertised is a self-contained JSON object using the following values.

Table 6 — AES-GMAC Algorithm Capabilities JSON Values

JSON Value	Description	JSON Type	Valid Values
algorithm	The MAC algorithm to be validated	string	"ACVP-AES-GMAC"
revision	The algorithm testing revision to use	string	"1.0"
prereqVals	Prerequistie algorithm validations	array of prereqAlgVal objects	See Section 4.2
direction	The GMAC direction(s) to test	array	"encrypt", "decrypt"
keyLen	The keyLen supported	array	128, 192, 256
ivLen	The IV lengths supported in bits, values **MUST** be mod 8	domain	Min: 8, Max: 1024, Inc: 8
ivGen	The IV generation method (from the perspective of the IUT)	string	"internal", "external"
ivGenMode	The IV generation mode for use when the IUT is internally generating IVs	string	"8.2.1", "8.2.2"
aadLen	The additional authenticated data length in bits	domain	Min: 0, Max: 65536, Inc: 8
tagLen	The supported mac/tag lengths	array	32, 64, 96, 104, 112, 120, 128

7.1.1. Example AES-GMAC Capabilities JSON Object

The following is an example JSON object advertising support for AES-GMAC.

```
{
   "algorithm": "ACVP-AES-GMAC",
   "revision": "1.0",
   "direction": [
     "encrypt",
     "decrypt"
   ],
   "keyLen": [
     128
   ],
```

```
    "ivLen": [
      96,
      120
    ],
    "ivGen": "external",
    "aadLen": [
      8,
      120
    ],
    "tagLen": [
      32,
      128
    ]
  }
```

Figure 4

8. HMAC

8.1. HMAC Algorithm Capabilities Registration

Each algorithm capability advertised is a self-contained JSON object using the following values.

Table 7 — HMAC Algorithm Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	The MAC algorithm and mode to be validated	string	See Section 8.2
revision	The algorithm testing revision to use	string	"1.0"
prereqVals	prerequistie algorithm validations	array of prereqAlgVal objects	See Section 4.2
keyLen	The keyLen Domain supported by the IUT in bits	domain	Min: 0, Max: 524288
macLen	The supported mac sizes, maximum is dependent on algorithm, see Section 8.2	domain	Min: 32

'keyLen' for HMAC contains a Domain of values, the server **MAY** choose values defined by these rules:

- 2 values below the Hash's block length. See Section 8.2
- The Hash's block length.
- 2 values above the Hash's block length.

'macLen' for HMAC contains a Domain of values, the server **MAY** choose values defined by these rules:

- The smallest HMAC length supported
- A second HMAC length supported
- The largest HMAC length supported

8.2. Supported HMAC Algorithms

The following HMAC algorithms contain specific individual properties:

Table 8 — Algorithms w/ block size and max MAC length.

Algorithm Value	Block Length	Max MAC Length
HMAC-SHA-1	512	160
HMAC-SHA2-224	512	224
HMAC-SHA2-256	512	256
HMAC-SHA2-384	1024	384

Algorithm Value	Block Length	Max MAC Length
HMAC-SHA2-512	1024	512
HMAC-SHA2-512/224	1024	224
HMAC-SHA2-512/256	1024	256
HMAC-SHA3-224	1152	224
HMAC-SHA3-256	1088	256
HMAC-SHA3-384	832	384
HMAC-SHA3-512	576	512

8.2.1. Example HMAC Capabilities JSON Object

The following is an example JSON object advertising support for HMAC.

```
{
  "algorithm": "HMAC-SHA-1",
  "revision": "1.0",
  "keyLen": [
    {
      "min": 8,
      "max": 2048,
      "increment": 8
    }
  ],
  "macLen": [
    {
      "min": 80,
      "max": 160,
      "increment": 8
    }
  ]
}
```

Figure 5

9. 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 HMAC, CMAC, and GMAC 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 9 — 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 10	array

An example of this would look like this

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

Figure 6

10. 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 session would require multiple test vector sets to be downloaded and processed by the ACVP client. Each test vector set represents an individual crypto algorithm, such as HMAC-SHA-1, HMAC-SHA2-224, CMAC-AES, etc. This section describes the JSON schema for a test vector set used with MAC crypto 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 10 — MAC Vector Set JSON Object

JSON Value	Description	JSON type
acvVersion	Protocol version identifier	string
vsId	Unique numeric identifier for the vector set	integer
algorithm	The algorithm used for the test vectors	string
revision	The algorithm testing revision to use	string
testGroups	Array of test group JSON objects, which are defined in Section 10.1.1, Section 10.2.1, Section 10.3.1, or Section 10.4.1 depending on the algorithm	array

10.1. CMAC-AES Test Vectors

10.1.1. CMAC-AES Test Groups JSON Schema

The testGroups element at the top level in the test vector JSON object is an array of test groups. Test vectors are grouped into similar test cases to reduce the amount of data transmitted in the vector set. For instance, all test vectors that use the same key size would be grouped together. The Test Group JSON object contains meta data that applies to all test vectors within the group. The following table describes the secure CMAC-AES JSON elements of the Test Group JSON object.

Table 11 — CMAC-AES Test Group JSON Object

JSON Value	Description	JSON type
tgId	Numeric identifier for the test group, unique across the entire vector set	integer
testType	Test category type	string
direction	The direction of the tests — gen or ver	string
keyLen	Length of key in bits to use	integer
msgLen	Length of message in bits	integer

JSON Value	Description	JSON type
macLen	Length of MAC in bits to generate/verify	integer
tests	Array of individual test vector JSON objects, which are defined in Section 10.1.2	array

10.1.2. CMAC-AES Test Case JSON Schema

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

Table 12 — CMAC-AES 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	hex
msg	Value of the message	hex
mac	MAC value, for CMAC verify	hex

10.1.3. Example CMAC-AES Test Vector JSON Object

The following is an example JSON test vector object for CMAC-AES, truncated for brevity.


```
{
    "vsId": 1,
    "algorithm": "CMAC-AES",
    "revision": "1.0",
    "testGroups": [{
            "tgId": 4,
            "testType": "AFT",
            "direction": "gen",
            "keyLen": 128,
            "msgLen": 2752,
            "macLen": 64,
            "tests": [{
                    "tcId": 25,
                    "key": "E2547E38B28B2C24892C133FF4770688",
                    "message": "89DE09D747FB4B2669B59759..."
                },
                {
                    "tcId": 26,
                    "key": "D1D99979CD96C3401291905F53B2ECA4",
                    "message": "6D054A7D1CD161B21F80E658..."
                },
                {
                    "tcId": 27,
                    "key": "672556E105B83BF39FC9E45268BD35D1",
```

```
                    "message": "39A3DD0652483A26FB9D71F3..."
                }
            ]
        },
        {
            "tgId": 10,
            "testType": "AFT",
            "direction": "ver",
            "keyLen": 128,
            "msgLen": 0,
            "macLen": 64,
            "tests": [{
                    "tcId": 73,
                    "key": "D5E09A89B2A4627BF987517B66A51564",
                    "message": "",
                    "mac": "CBBC968859633C24"
                },
                {
                    "tcId": 74,
                    "key": "646A150116ABAA37662FE9D8BB278693",
                    "message": "",
                    "mac": "21D069331196E579"
                },
                {
                    "tcId": 75,
                    "key": "5C185C01FBC57A40FC373F199374D1CC",
                    "message": "",
                    "mac": "9507F00153543DE6"
                }
            ]
        }
    ]
}
```


Figure 7

10.2. CMAC-TDES Test Vectors

10.2.1. CMAC-TDES Test Groups JSON Schema

The testGroups element at the top level in the test vector JSON object is an array of test groups. Test vectors are grouped into similar test cases to reduce the amount of data transmitted in the vector set. For instance, all test vectors that use the same key size would be grouped together. The Test Group JSON object contains meta data that applies to all test vectors within the group. The following table describes the secure CMAC-TDES JSON elements of the Test Group JSON object.

Table 13 — CMAC-TDES Test Group JSON Object

JSON Value	Description	JSON type
tgId	Numeric identifier for the test group, unique across the entire vector set	integer
testType	Test category type	string
direction	The direction of the tests — gen or ver	string
keyLen	Length of key in bits to use	integer
msgLen	Length of message in bits	integer
macLen	Length of MAC in bits to generate/verify	integer
tests	Array of individual test vector JSON objects, which are defined in Section 10.2.2	array

10.2.2. CMAC-TDES Test Case JSON Schema

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

Table 14 — CMAC-TDES Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
key1, key2, key3	Encryption keys to use for TDES	hex
msg	Value of the message	hex
mac	MAC value, for CMAC verify	hex

10.2.3. Example CMAC-TDES Test Vector JSON Object

The following is an example JSON test vector object for CMAC-TDES, truncated for brevity.


```
{
    "vsId": 1,
    "algorithm": "CMAC-TDES",
    "revision": "1.0",
    "testGroups": [{
            "tgId": 4,
            "testType": "AFT",
            "direction": "gen",
            "keyLen": 192,
            "msgLen": 752,
            "macLen": 32,
            "keyingOption": 1,
            "tests": [{
                    "tcId": 25,
                    "message": "4D2D07AE0224289BE111...",
                    "key1": "7FFDB31A3A06BF0D",
```

```
                "key2": "5201FFE83CC5253E",
                "key3": "BBECA60F1D631BB8"
            },
            {
                "tcId": 26,
                "message": "7A70F3CE17598AC3553F...",
                "key1": "82E572E29C84009E",
                "key2": "28390918155E4891",
                "key3": "A0D24C94C306FE56"
            },
            {
                "tcId": 27,
                "message": "2DAF1E213A6AE9E88FEB...",
                "key1": "662A89E47D86F6A0",
                "key2": "CB9D612E199B4AD2",
                "key3": "AE3AEBA19FAB4340"
            }
        ]
    },
    {
        "tgId": 10,
        "testType": "AFT",
        "direction": "ver",
        "keyLen": 192,
        "msgLen": 0,
        "macLen": 32,
        "keyingOption": 1,
        "tests": [{
                "tcId": 73,
                "message": "",
                "mac": "D78010B3",
                "key1": "D127F902CFF69A82",
                "key2": "785D99EF117E5B56",
                "key3": "36D4CB2C34A3AF30"
            },
            {
                "tcId": 74,
                "message": "",
                "mac": "5313D9E8",
                "key1": "9D5DAC4D75E0A349",
                "key2": "6A55855C62D8C767",
                "key3": "CBD9FC7CCBDA8BE9"
            },
            {
                "tcId": 75,
                "message": "",
```

```
                        "mac": "07B1926F",
                        "key1": "B711BA268A1F3663",
                        "key2": "AA83B2195E85BFC6",
                        "key3": "AA1CFBC128AAE0FD"
                    }
                ]
            }
        ]
}
```


Figure 8

10.3. AES-GMAC Test Vectors

10.3.1. AES-GMAC Test Groups JSON Schema

The testGroups element at the top level in the test vector JSON object is an array of test groups. Test vectors are grouped into similar test cases to reduce the amount of data transmitted in the vector set. For instance, all test vectors that use the same key size would be grouped together. The Test Group JSON object contains meta data that applies to all test vectors within the group. The following table describes the secure GMAC JSON elements of the Test Group JSON object.

Table 15 — AES-GMAC Test Group JSON Object

JSON Value	Description	JSON Type
tgId	Numeric identifier for the test group, unique across the entire vector set	integer
testType	Test category type	string
direction	The direction of the tests — encrypt or decrypt	string
keyLen	Length of key in bits to use	integer
ivLen	Length of IV in bits	integer
ivGen	IV Generation (internal or external)	string
ivGenMode	The mode an internal IV has been generated using	string
aadLen	Length of AAD in bits	integer
tagLen	Length of tag/MAC in bits to generate/verify	integer
tests	Array of individual test vector JSON objects, which are defined in Section 10.3.2	array

10.3.2. AES-GMAC Test Case JSON Schema

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

Table 16 — AES-GMAC 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	hex
iv	Value of the IV	hex
aad	Value of the AAD	hex
tag	MAC/tag value, for validating on a decrypt operation	hex

10.3.3. Example AES-GMAC Test Vector JSON Object

The following is an example JSON test vector object for AES-GMAC, truncated for brevity


```
{
  "vsId": 0,
  "algorithm": "ACVP-AES-GMAC",
  "revision": "1.0",
  "testGroups": [{
      "tgId": 1,
      "testType": "AFT",
      "direction": "encrypt",
      "keyLen": 128,
      "ivLen": 96,
      "ivGen": "external",
      "aadLen": 0,
      "tagLen": 32,
      "tests": [{
        "tcId": 1,
        "key": "B1E9747F9E016F0376B1F379CD345B8A",
        "aad": "",
        "iv": "6294CEEDFCC3037A023100E8"
      }]
    },
    {
      "tgId": 2,
      "testType": "AFT",
      "direction": "decrypt",
      "keyLen": 128,
      "ivLen": 96,
      "ivGen": "external",
      "aadLen": 0,
      "tagLen": 32,
      "tests": [{
        "tcId": 2,
        "key": "CD345B8AE016F03765E9747F9B1F379A",
```

```
            "aad": "",
            "iv": "6207CEEDFA094CC3032310E8",
            "tag": "AB1254CE"
          }]
        }
      ]
    }
```


Figure 9

10.4. HMAC Test Vectors

10.4.1. HMAC Test Groups JSON Schema

The testGroups element at the top level in the test vector JSON object is an array of test groups. Test vectors are grouped into similar test cases to reduce the amount of data transmitted in the vector set. For instance, all test vectors that use the same key size would be grouped together. The Test Group JSON object contains meta data that applies to all test vectors within the group. The following table describes the secure HMAC JSON elements of the Test Group JSON object.

Table 17 — HMAC Test Group JSON Object

JSON Value	Description	JSON type
tgId	Numeric identifier for the test group, unique across the entire vector set	integer
testType	Test category type	string
keyLen	Length of key in bits to use	integer
msgLen	Length of message in bits	integer
macLen	Length of MAC in bits to generate	integer
tests	Array of individual test vector JSON objects, which are defined in Section 10.4.2	array

10.4.2. HMAC Test Case JSON Schema

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

Table 18 — HMAC Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
key	The value of the key	hex
msg	Value of the message	hex

10.4.3. Example HMAC Test Vector JSON Object

The following is an example JSON test vector object for HMAC, truncated for brevity.


```
{
    "vsId": 1,
    "algorithm": "HMAC-SHA-1",
    "revision": "1.0",
    "testGroups": [{
        "tgId": 1,
        "testType": "AFT",
        "keyLen": 56,
        "msgLen": 128,
        "macLen": 80,
        "tests": [{
                "tcId": 1,
                "key": "0CBB3AA866E4D1",
                "msg": "28CD4091D45F28CD5709CC9B6F1E9D0D"
            },
            {
                "tcId": 2,
                "key": "7FB3F60ACB9FB7",
                "msg": "9F224BF653F9BE143FFA0518D12761F7"
            },
            {
                "tcId": 3,
                "key": "3834463234DA39",
                "msg": "F0FA740D261D5916B06F09AFBB04C94E"
            }
        ]
    }]
}
```


Figure 10

11. Responses

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

Table 19 — Response JSON Object

JSON Property	Description	JSON Type
acvVersion	The ACVP version used	string
vsId	The vector set identifier	integer
testGroups	The test group objects in the response, see Table 20	array

An example of this is the following

```
{
 "acvVersion": "version",
 "vsId": 1,
 "testGroups": [ ... ]
}
```

Figure 11

The 'testGroups' section is used to organize the ACVP client response in a similar manner to how it distributes vectors.

Table 20 — Response Group Objects

JSON Property	Description	JSON Type
tgId	The test group identifier	integer
tests	The test case objects in the response, depending on the algorithm see Table 21, Table 22, Table 23 or Table 24	array

An example of this is the following

```
{
 "tgId": 1,
 "tests": [ ... ]
}
```

Figure 12

11.1. CMAC-AES Test Vector Responses

11.1.1. CMAC-AES Vector Set Group Responses

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

Table 21 — CMAC-AES Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
mac	The value of the computed MAC output for 'gen' test type groups	hex
testPassed	The result of CMAC verify for 'ver' test type groups	boolean

11.1.2. Example CMAC-AES Test Vector Response JSON Object

The following is an example JSON test vector response object for CMAC-AES.


```
{
    "vsId": 1,
    "testGroups": [{
            "tgId": 4,
            "tests": [{
                    "tcId": 25,
                    "mac": "7AA2D56A0AE76620"
                },
                {
                    "tcId": 26,
                    "mac": "9E23524D3CD18C93"
                },
                {
                    "tcId": 27,
                    "mac": "4D51872CBFAA102A"
                }
            ]
        },
        {
            "tgId": 10,
            "tests": [{
                    "tcId": 73,
                    "testPassed": true
                },
                {
                    "tcId": 74,
```

```
                        "testPassed": true
                    },
                    {
                        "tcId": 75,
                        "testPassed": true
                    }
                ]
            }
        ]
}
```


Figure 13

11.2. CMAC-TDES Test Vector Responses

11.2.1. CMAC-TDES Vector Set Group Responses

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

Table 22 — CMAC-TDES Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
mac	The value of the computed MAC output for ‘gen’ test type groups	hex
testPassed	The result of CMAC verify for ‘ver’ test type groups	boolean

11.2.2. Example CMAC-TDES Test Vector Response JSON Object

The following is an example JSON test vector response object for CMAC-TDES.


```
{
    "vsId": 1,
    "algorithm": "CMAC-TDES",
    "revision": "1.0",
    "testGroups": [{
            "tgId": 4,
            "tests": [{
                        "tcId": 25,
                        "mac": "6FA27FDC"
                    },
                    {
                        "tcId": 26,
                        "mac": "89CE2842"
                    },
```

```
                {
                    "tcId": 27,
                    "mac": "5E03D980"
                }
            ]
        },
        {
            "tgId": 10,
            "tests": [{
                    "tcId": 73,
                    "testPassed": true
                },
                {
                    "tcId": 74,
                    "testPassed": true
                },
                {
                    "tcId": 75,
                    "testPassed": true
                }
            ]
        }
    ]
}
```


Figure 14

11.3. AES-GMAC Test Vector Responses

11.3.1. AES-GMAC Vector Set Group Responses

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

Table 23 — AES-GMAC Test Case Results JSON Object

JSON Value	Description	JSON Type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
tag	Value of the computed tag/MAC output, for 'encrypt' direction groups	hex
testPassed	The result of decrypt verify, for 'decrypt' direction groups	boolean

11.3.2. Example AES-GMAC Test Vector Response JSON Object

The following is an example JSON test vector response object for AES-GMAC.

```
{
 "vsId": 1,
 "algorithm": "ACVP-AES-GMAC",
 "revision": "1.0",
 "testGroups": [{
     "tgId": 1,
     "tests": [{
       "tcId": 1,
       "tag": "6FA27FDC"
     }]
   },
   {
     "tgId": 2,
     "tests": [{
       "tcId": 2,
       "testPassed": true
     }]
   }
 ]
}
```


Figure 15

11.4. HMAC Test Vector Responses

11.4.1. HMAC Vector Set Group Responses

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

Table 24 — HMAC Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
mac	Value of the computed MAC output	hex

11.4.2. Example HMAC Test Vector Response JSON Object

The following is an example JSON test vector response object for HMAC.


```
{
    "vsId": 1,
    "algorithm": "HMAC-SHA-1",
    "revision": "1.0",
    "testGroups": [{
        "tgId": 1,
```

```
        "tests": [{
                "tcId": 1,
                "mac": "0970D053D6829C251070"
            },
            {
                "tcId": 2,
                "mac": "3A476E0407D7ADF14792"
            },
            {
                "tcId": 3,
                "mac": "5B219B4C4862242DB175"
            }
        ]
    }]
}
```

Figure 16

12. Security Considerations

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

Appendix A — Terminology

For the purposes of this document, the following terms and definitions apply.

A.1.

Prompt
JSON sent from the server to the client describing the tests the client performs
Registration
The initial request from the client to the server describing the capabilities of one or several algorithm, mode and revision combinations
Response
JSON sent from the client to the server in response to the prompt
Test Case
An individual unit of work within a prompt or response
Test Group
A collection of test cases that share similar properties within a prompt or response
Test Vector Set
A collection of test groups under a specific algorithm, mode, and revision
Validation
JSON sent from the server to the client that specifies the correctness of the response

Appendix B — Abbreviations and Acronyms

ACVP	Automated Crypto Validation Protocol
JSON	Javascript Object Notation

Appendix C — Revision History

Table C-1

Version	Release Date	Updates
1	2018-08-01	Initial Release

Appendix D — 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.

P. Hoffman (December 2016) *The "xml2rfc" Version 3 Vocabulary* (Internet Engineering Task Force), RFC 7991, December 2016. RFC 7991. RFC RFC7991. DOI 10.17487/RFC7991. https://www.rfc-editor.org/info/rfc7991.

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 (July 2008) *The Keyed-Hash Message Authentication Code (HMAC)* (Gaithersburg, MD), July 2008. FIPS 198-1. https://doi.org/10.6028/NIST.FIPS.198-1.

Morris J. Dworkin (May 2005 (updated October 2016)) *Recommendation for Block Cipher Modes of Operation — the CMAC Mode for Authentication* (Gaithersburg, MD), May 2005 (updated October 2016). SP 800-38B. https://doi.org/10.6028/NIST.SP.800-38B.

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.

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.

Keller S (2011) *The CMAC Validation System (CMACVS)* (National Institute of Standards and Technology, Gaithersburg, MD), 2011.

Bassham III L (2016) *The Keyed-Hash Message Authentication Code Validation System (HMACVS)* (National Institute of Standards and Technology, Gaithersburg, MD), 2016.

Bassham III L (2016) *The Secure Hash Algorithm 3 Validation System (SHA3VS)* (National Institute of Standards and Technology, Gaithersburg, MD), 2016.

Bassham III L (2014) *The Secure Hash Algorithm Validation System (SHAVS)* (National Institute of Standards and Technology, Gaithersburg, MD), 2014.