ACVP SP800-108 Key Derivation Function JSON Specification

Christopher Celi
Information Technology Laboratory
Computer Security Division

June 05, 2019

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

Abstract

This document defines the JSON schema for testing SP 800-108 KDF 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 SP 800-108 KDF implementations using ACVP.

Disclaimer

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

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 SP 800-108 KDF implementations using ACVP.

2. Supported SP 800-108 KDFs

The following key derivation functions **MAY** be advertised by the ACVP compliant cryptographic module:

- KDF / 1.0

3. Test Types and Test Coverage

This section describes the design of the tests used to validate SP 800-108 KDF implementations.

3.1. Test Types

There is only one test type: functional tests. Each has a specific value to be used in the testType field. The testType field definition is:

- "AFT"—Algorithm Functional Test. These tests can be processed by the client using a normal 'derive_key' operation. AFTs cause the implementation under test to exercise normal operations on a single block, multiple blocks, or partial blocks. In all cases, random data is used. The functional tests are designed to verify that the logical components of the key deriviation process are operating correctly.

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [SP 800-108].

3.2.1. Requirements Covered

- The ACVP server tests the IUT's ability to derive keying material using the "modes of iteration" defined in sections 5.1, 5.2 and 5.3 of [SP 800-108]. The server supports testing the IUT against the various MACs or PRFs listed in Section 4.3.3. It also supports testing varying 1) the location of the counter within the input data, 2) the length of the derived keying material, and 3) the counter length.

3.2.2. Requirements Not Covered

- The tests described in this document do not validate the construction of the fixed input data string described in [SP 800-108] Sections 5, 7.5 and 7.6.

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 SP 800-108 KDF 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 SP 800-108 KDF Validations

Each SP800-108 KDF implementation relies on other cryptographic primitives. For example, the KDF must utilize an underlying MAC 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 — SP800-108 KDF Required Prerequisites

JSON Value	Description	JSON Type	Valid Values
algorithm	A prerequisite algorithm	string	AES, DRBG, CMAC, HMAC
valValue	Algorithm validation number	string	Actual number or “same”
prereqAlgVal	Prerequisite algorithm validation	object with algorithm and valValue properties	See above
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See above

4.3. Property Registration

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

A registration **SHALL** use these properties:

Table 3 — Registration Properties

JSON Property	Description	JSON Type	Valid Values
algorithm	The KDF to be validated.	string	KDF
revision	The algorithm testing revision to use.	string	“1.0”
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See Section 4.2
capabilities	Array of JSON objects, each with fields pertaining to the KDF mode identified uniquely by the combination of the “kdfMode” and indicated properties	array of JSON objects	See Section 4.3.2

4.3.1. Supported SP 800-108 KDF Modes

The following SP800-108 KDF modes or “modes of iteration” may be advertised by the ACVP compliant crypto module:

- counter
- feedback
- double pipeline iteration

4.3.2. Supported KDF Modes Capabilities

The KDF mode capabilities are advertised as JSON objects within the ‘capabilities’ value of the ACVP registration message—see Table 2. The ‘capabilities’ value is an array, where each array element is a JSON object corresponding to a particular KDF mode defined in this section. The ‘capabilities’ value is part of the ‘capability_exchange’ element of the ACVP JSON registration message. See the ACVP specification [ACVP] for details on the registration message.

Each KDF mode’s capabilities are advertised as JSON objects.

The complete list of KDF key generation capabilities may be advertised by the ACVP compliant crypto module:

Table 4 — KDF Mode Capabilities

JSON Value	Description	JSON Type	Valid Values
kdfMode	The type of SP800-108 KDF or “mode of iteration”	string	See Section 4.3.1
macMode	The MAC or PRF algorithm used	string	See Section 4.3.3
supportedLengths	The supported derived keying material lengths in bits	domain	Min: 1, Max: 4096
fixedDataOrder	Describes where the counter appears in the fixed data	array	Any non-empty subset of {”none”, “after fixed data”, “before fixed data”, “middle fixed data”, “before iterator”}
counterLength	The length of the counter in bits	array	Any non-empty subset of {0, 8, 16, 24, 32}
supportsEmptyIv	Whether or not the IUT supports an empty IV	boolean	true/false

JSON Value	Description	JSON Type	Valid Values
requiresEmptyIv	Whether or not the IUT requires an empty IV	boolean	true/false
customKeyInLength	Optional value used to control the length of the keyIn produced by the ACVP server for the capability. This field cannot be used to alter the keyIn length for AES/TDES based macModes, as the keyIns expected by those algorithms is fixed.	integer	112-4096
NOTE 1 – The ‘fixedDataOrder’ options “none” and “before iterator” are not valid for “counter” KDF. The ‘fixedDataOrder’ option “middle fixed data” is not valid for “feedback” nor “double pipeline iteration” KDF. NOTE 2 – A ‘counterLength’ of 0 describes that there is no counter used. The 0 option is not valid for “counter” KDF. NOTE 3 – When ‘counterLength’ contains a value of “0”, ‘fixedDataOrder’ must contain a value of “none” and vice versa.			

4.3.3. Supported SP 800-108 KDF MACs

The following MAC or PRF functions **MAY** be advertised by an ACVP compliant client

- CMAC-AES128
- CMAC-AES192
- CMAC-AES256
- CMAC-TDES
- HMAC-SHA1
- 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

4.4. Registration Example

The following is a example JSON object advertising support for a SP 800-108 KDF.

{ "algorithm": "KDF", "revision": "1.0", "prereqVals": [{ "algorithm": "SHA", "valValue": "123456" }, { "algorithm": "DRBG", "valValue": "123456" }], "capabilities": [{ "kdfMode": "counter", "macMode": ["CMAC-AES128", "CMAC-AES192", "CMAC-AES256", "CMAC-TDES", "HMAC-SHA-1", "HMAC-SHA2-224", "HMAC-SHA2-256", "HMAC-SHA2-384", "HMAC-SHA2-512"], "supportedLengths": [{ "min": 8, "max": 1024, "increment": 1 }], "fixedDataOrder": ["after fixed data", "before fixed data", "middle fixed data"], "counterLength": [8, 16, 24, 32], "supportsEmptyIv": false }, { "kdfMode": "feedback", "macMode": ["CMAC-AES128", "CMAC-AES192", "CMAC-AES256", "CMAC-TDES", "HMAC-SHA-1", "HMAC-SHA2-224", "HMAC-SHA2-256", "HMAC-SHA2-384", "HMAC-SHA2-512"], "supportedLengths": [{ "min": 8, "max": 1024, "increment": 1 }], "fixedDataOrder": ["none", "after fixed data", "before fixed data", "before iterator"], "counterLength": [0, 8, 16, 24, 32], "supportsEmptyIv": true, "requiresEmptyIv": false }, { "kdfMode": "double pipeline iteration", "macMode": ["CMAC-AES128", "CMAC-AES192", "CMAC-AES256", "CMAC-TDES", "HMAC-SHA-1", "HMAC-SHA2-224", "HMAC-SHA2-256", "HMAC-SHA2-384", "HMAC-SHA2-512"], "supportedLengths": [{ "min": 8, "max": 1024, "increment": 1 }], "fixedDataOrder": ["none", "after fixed data", "before fixed data", "before iterator"], "counterLength": [0, 8, 16, 24, 32], "supportsEmptyIv": false }] }

Figure 2

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 SP 800-108 KDF 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 5 — 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 3

5.1. 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 SP 800-108 KDF JSON elements of the Test Group JSON object.

Table 6 — Test Group JSON Object

JSON Values	Description	JSON Type
tgId	Test group identifier	integer
kdfMode	The kdfMode used for the test group	string
macMode	Psuedorandom function (PRF) HMAC or CMAC used	string
counterLocation	"none", "after fixed data", "before fixed data", "middle fixed data", or "before iterator"	string
keyOutLength	Expected length of the derived keying material or key in bits	integer
counterLength	Expected length of the counter in bits	integer
zeroLengthIv	Whether or not the group utilizes a null IV	boolean
testType	Describes the operation being performed	string
tests	Array of individual test cases	array
NOTE – The feedback mode KDF counter location of "counter after iterator" is referenced with the option "before fixed data" in the 'counterLocation' specification.		

The 'tgId', 'testType' and 'tests' objects **MUST** appear in every test group element communicated from the server to the client as a part of a prompt. Other properties are dependent on which 'testType' the group is addressing.

5.2. 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 SP 800-108 KDF test vector.

Table 7 — Test Case JSON Object

JSON Values	Description	JSON Type
tcId	Test case idenfitier	integer
keyIn	Input key	hex
iv	The initialization vector used only for feedback KDFs	hex

JSON Values	Description	JSON Type
deferred	Indicates the client is required to provide additional input values to complete the test case	boolean
NOTE 1 – If the “middle fixed data” option is used for the ‘counterLocation’, then the test case is deferred from the server. In this case, the client **MUST** provide a ‘breakLocation’ integer value which is the length of the fixed data before the counter is inserted. NOTE 2 – If the ‘deferred’ property is not present, it **MAY** be assumed to be ‘false’.		

5.3. Example Test Vector JSON Object

The following is a example JSON object for SP 800-108 KDF test vectors sent from the ACVP server to the crypto module.

[{ "acvVersion": "0..54" }, { "vsId": 1564, "algorithm": "counterMode", "revision": "1.0", "testGroups": [{ "tgId": 1, "kdfMode": "counter", "macMode": "CMAC-AES128", "counterLocation": "after fixed data", "keyOutLength": 1024, "counterLength": 8, "tests": [{ "tcId": 1, "keyIn": "5DA38931E8D9174BC3279C8942D2DB82", "deferred": false }, { "tcId": 2, "keyIn": "58F5426A40E3D5D2C94F0F97EB30C739", "deferred": false }] }] }]

Figure 4

6. Test Vector 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 8 — Vector Set Response Properties

JSON Property	Description	JSON Type
acvVersion	The version of the protocol	string
vsId	The vector set identifier	integer
testGroups	The test group data	array

An example of this is the following

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

Figure 5

The testGroups 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 their response. This structure helps accommodate that.

Table 9 — Vector Set Group Response Properties

JSON Property	Description	JSON Type
tgId	The test group identifier	integer
tests	The test case data	array

An example of this is the following

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

Figure 6

The testCase 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 their response. This structure helps accommodate that.

The following table describes the JSON properties that represent a test case response for a SP 800-108 KDF.

Table 10 — Test Case Results JSON Properties

JSON Property	Description	JSON Type
tcId	The test case identifier	integer
breakLocation	The bit location in the fixed data where the counter is placed	integer
fixedData	The fixed input data used by the IUT	hex
keyOut	The outputted keying material or key	hex
NOTE – the fixedData or fixed input data string that is used by the IUT is needed by the ACVP server to verify that the IUT correctly derived the keying material. The server does not validate the correct construction of the fixed input data string. For guidance on constructing a valid fixed input data string, please consult [SP 800-108] Sections 5, 7.5 and 7.6.		

6.1. Example Test Vector Response JSON

The following is an abbreviated example of a JSON object for SP800-108 KDF test results sent from the crypto module to the ACVP server.

[{ "acvVersion": <acvp-version> }, { "vsId": 1564, "testGroups": [{ "tgId": 1, "tests": [{ "tcId":
1, "keyOut":
"94D58F22FA9092B0375F7EE6841B6775226703E3232BF9CF496E4EF3CDE1037765DDC060C08C9B3A84
"fixedData": "FBF14DF02EE6C7DABCA6EF9AF59BB9A2" }] }] }]

Figure 7

NOTE – Please note that the values used in the example JSON object are not real values. In particular, the value for the fixedData property is not an example of a validly formed fixed input data string.

7. 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	2019-06-05	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.

Lily Chen (October 2009) *Recommendation for Key Derivation Using Pseudorandom Functions (Revised)* (Gaithersburg, MD), October 2009. SP 800-108. https://doi.org/10.6028/NIST.SP.800-108.

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.