ACVP IKEv1 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 SP800-135 IKEv2 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 SP800-135 IKEv2 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 SP800-135 IKEv2 KDF implementations using ACVP.

2. Supported KDFs

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

- kdf-components / ikev2

3. Test Types and Test Coverage

This section describes the design of the tests used to validate SP800-135 IKEv2 KDF implementations. There is only one test type: functional 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 '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.

3.1. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to XXX.

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 IKEv2 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. Property Registration

The IKEv2 KDF mode capabilities are advertised as JSON objects within the 'capabilities_exchange' property.

4.3. Registration Example

A registration **SHALL** use these properties

Table 2 — IKEv2 KDF Mode Capabilities JSON Values

JSON Value	Description	JSON Type	Valid Values
algorithm	Name of the algorithm to be validated	string	"kdf-components"
mode	Mode of the algorithm to be validated	string	"ikev2"
revision	ACVP Test version	string	"1.0"
capabilities	Array of objects describing capabilities	array	Contains each of the below properties
initiatorNonceLength	The supported initiator nonce lengths used by the IUT in bits	domain	Min: 64, Max: 2048
responderNonceLength	The lengths of data the IUT supports in bits	domain	Min: 64, Max: 2048
diffieHellmanSharedSecretLength	The lengths of Diffie Hellman shared secrets the IUT supports in bits	domain	Min: 224, Max: 8192
derivedKeyingMaterialLength	The lengths of derived key material the IUT supports in bits	domain	Min: 160, Max: 16384

JSON Value	Description	JSON Type	Valid Values
hashAlg	Valid hash algorithms used by the IUT	array	See Section 4.3.1

An example registration within an algorithm capability exchange looks like this


```
"capability_exchange":
[
    {
        "algorithm": "kdf-components",
        "mode": "IKEv2",
        "revision": "1.0",
        "capabilities": [
            {
                "hashAlg": [
                    "sha-1",
                    "sha2-224",
                    "sha2-256"
                ],
                "initiatorNonceLength": [
                    {
                        "min": 64,
                        "max": 2048,
                        "increment": 1
                    }
                ],
                "responderNonceLength": [
                    {
                        "min": 64,
                        "max": 2048,
                        "increment": 1
                    }
                ],
                "diffieHellmanSharedSecretLength": [
                    {
                        "min": 224,
                        "max": 8192,
                        "increment": 1
                    }
                ],
                "derivedKeyingMaterialLength": [
                    {
                        "min": 384,
                        "max": 16384,
```

```
                    "increment": 1
                }
            ]
        },
        {
            "hashAlg": [
                "sha2-384",
                "sha2-512"
            ],
            "initiatorNonceLength": [
                {
                    "min": 64,
                    "max": 2048,
                    "increment": 1
                }
            ],
            "responderNonceLength": [
                {
                    "min": 64,
                    "max": 2048,
                    "increment": 1
                }
            ],
            "diffieHellmanSharedSecretLength": [
                {
                    "min": 224,
                    "max": 8192,
                    "increment": 1
                }
            ],
            "derivedKeyingMaterialLength": [
                {
                    "min": 1024,
                    "max": 16384,
                    "increment": 1
                }
            ]
        }
    ]
  }
]
```


Figure 2

4.3.1. Valid Hash Functions

The following hash functions **MAY** be advertised by an ACVP compliant client under the ‘hashAlg’ property

- SHA-1
- SHA2-224
- SHA2-256
- SHA2-384
- SHA2-512

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 SP800-135 IKEv2 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 3 — 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

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 SP800-135 IKEv2 KDF JSON elements of the Test Group JSON object

Table 4 — Test Group JSON Object

JSON Value	Description	JSON Type
tgId	Test group identifier	integer
testType	Test operations to be performed	string
hashAlg	The SHA type used for the test vectors	string
nInitLength	Length of initiator nonce in bits	hex
nRespLength	Length of responder nonce in bits	hex
derivedKeyingMaterialLength	Derived Key Matrerial length expected in bits	integer
dhLength	Diffie Hellman shared secret length in bits	integer
tests	Array of individual test cases	array

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' (see Section 3) the group is addressing.

5.2. Test Cases

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 SP800-135 IKEv2 KDF test vector.

Table 5 — Test Case JSON Object

JSON Value	Description	JSON Type
tcId	Test case idenfitier	integer
nInit	Value of the initiator nonce	hex
nResp	Value of the responder nonce	hex
gir	Diffie-Hellman shared secret	hex
girNew	New Diffie-Hellman shared secret	hex
spiInit	security parameter indice of the initiator	hex
spiResp	security parameter indice of the responder	hex

Here is an abbreviated yet fully constructed example of the prompt.


```
{
  "vsId": 1,
  "algorithm": "kdf-components",
  "mode": "IKEv2",
  "revision": "1.0",
```

```
  "testGroups": [
    {
      "tgId": 1,
      "hashAlg": "SHA-1",
      "dhLength": 224,
      "nInitLength": 64,
      "nRespLength": 2048,
      "derivedKeyingMaterialLength": 16384,
      "testType": "AFT",
      "tests": [
        {
          "tcId": 1,
          "nInit": "258A2A59B5A960A3",
          "nResp": "1BC7543704848EF6...",
          "gir": "9528B0F97999E1C7FE...",
          "girNew": "EC54C9B02FFAFEC...",
          "spiInit": "52D5397B0061602B",
          "spiResp": "E45E291943E3E5ED"
        },
        {
          "tcId": 2,
          "nInit": "9986940729199F59",
          "nResp": "479E9DC203FFE874...",
          "gir": "4380C15BC19F4872EF...",
          "girNew": "57D5AFAE6D80C15...",
          "spiInit": "C383DF2C6F9072BF",
          "spiResp": "5A7026194D4ACF79"
        }
      ]
    }
  ]
}
```


Figure 4

6. 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 6 — Vector Set Response JSON Object

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 7 — Vector Set Group Response JSON Object

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 following table describes the JSON object that represents a test case response for a SP800-135 IKEv2 KDF.

Table 8 — Test Case Results JSON Object

JSON Value	Description	JSON Type
tcId	The test case identifier	integer

JSON Value	Description	JSON Type
sKeySeed	Results of the extraction step	hex
sKeySeedReKey	Results of the newly created skeyid	hex
derivedKeyingMaterial	Derived key Material from expansion step	hex
derivedKeyingMaterialChild	Expansion step results for child SA	hex
derivedKeyingMaterialDh	Expansion step results for child SA DH	hex

Here is an abbreviated example of the response

```
{
  "vsId": 1,
  "algorithm": "kdf-components",
  "mode": "IKEv2",
  "revision": "1.0",
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "sKeySeed": "B5B0B203F931C2BD06D9...",
          "derivedKeyingMaterial": "C807976...",
          "derivedKeyingMaterialChild": "BA...",
          "derivedKeyingMaterialDh": "4A222...",
          "sKeySeedReKey": "7076E7DB098CE1A..."
        },
        {
          "tcId": 2,
          "sKeySeed": "DAF33468A586B7E705CA...",
          "derivedKeyingMaterial": "5659749...",
          "derivedKeyingMaterialChild": "2A...",
          "derivedKeyingMaterialDh": "4554A...",
          "sKeySeedReKey": "CF78ADF8EE17348..."
        }
      ]
    }
  ]
}
```

Figure 7

7. Security Considerations

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

8. Example SP800-135 IKEv2 KDF Capabilities JSON Object

The following is a example JSON object advertising support for SHA-256.

{ "algorithm": "kdf-components", "mode": "ikev2", "revision": "1.0", "prereqVals": [{ "algorithm": "SHA", "valValue": "123456" }, { "algorithm": "HMAC", "valValue": "123456" }], "capabilities": { "hashAlg": ["SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512"], "initiatorNonceLength": [{ "min": 64, "max": 2048, "increment": 1 }], "responderNonceLength": [{ "min": 64, "max": 2048, "increment": 1 }], "diffieHellmanSharedSecretLength": [{ "min": 224, "max": 8192, "increment": 1 }], "derivedKeyingMaterialLength": [{ "min": 512, "max": 16384, "increment": 1 }] } }

Figure 8

9. Example Test Vectors JSON Object

The following is a example JSON object for SP800-135 IKEv2 KDF test vectors sent from the ACVP server to the crypto module.

[{ "acvVersion": <acvp-version> }, { "vsId": 1564, "algorithm": "kdf-components", "mode": "ikev2", "revision": "1.0", "testGroups": [{ "tgId": 1, "hashAlg": "SHA-1", "nInitLength": 837, "nRespLength": 64, "dhLength": 8192, "derivedKeyingMaterialLength": 512, "tests": [{ "tcId": 1, "nInit":
"7608C5146A67DFA155DCD559F6F4771183035A8ACA913A0105922DEC57A79DBEFF0B5C0312EB7215
"nResp": "F3486687B23C20B4", "spiInit": "EEBB344F716A0421", "spiResp": "4EBA56C083C2366A", "gir":
"BC9505DF26B41AE3403D725D388FC679B753DBAD7F860E1F9229C2286833A248180C1CE061DF1C3F0
"girNew":
"F639A0018F3F2674796AE876850513194C29F970AA0C755C2897F89FEF85BFD31DE7B55A615D1F472B

Figure 9

10. Example Test Results JSON Object

The following is a example JSON object for SP800-135 IKEv2 KDF test results sent from the crypto module to the ACVP server.

[{ "acvVersion": <acvp-version> }, { "vsId": 1564, "testGroups": [{ "tgId": 1, "tests":
[{ "tcId": 1, "sKeySeed": "08F33F57DE075F1FC4033066BA42AA66E41B4CA7",
"derivedKeyingMaterialDh":
"527566D0861C294D92F633B6FDDF1CAC56AADAB92386D37126D12E6588EF9621C4EA1D33D20B1A9
"derivedKeyingMaterialChild":
"A5E3D4BA0C794393772C82229625621503F6E2007DD1199664B7D2AA42E5F380AA49D4F7D044DC256
"derivedKeyingMaterialDh":
"9D157EFE06B56A872BBF682168E2A3D6AA301C178540B4EE433AE44667C737C80B318BB9E4637CA2
"sKeySeedReKey": "04C03F09F09701151B7772D4196796EE08448EA8" }] }] }]

Figure 10

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.

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.