ACVP KDA OneStep Specification

Russell Hammett
HII Technical Solutions Division
302 Sentinel Drive, Suite #300, Annapolis Junction, MD 20701

December 11, 2020

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

Abstract

This document defines the JSON schema for testing KDA-OneStep SP800-56C 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 KDA-OneStep SP800-56C 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 KDA-OneStep SP800-56C implementations using ACVP.

2. Supported KDA OneStep

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

- KDA / OneStep / SP800-56Cr1
- KDA / OneStep / SP800-56Cr2

3. Test Types and Test Coverage

The ACVP server performs a set of tests on the KDAs specific to the KAS protocol in order to assess the correctness and robustness of the implementation. A typical ACVP validation session **SHALL** require multiple tests to be performed for every supported permutation of KDA capabilities. This section describes the design of the tests used to validate implementations of KDA algorithms.

3.1. Test Types

There are two test types for KDA testing:

- “AFT”—Algorithm Function Test. In the AFT test mode, the IUT **SHALL** act as a party in the Key Agreement with the ACVP server. The server **SHALL** generate and provide all necessary information for the IUT to perform a successful key agreement; both the server and IUT **MAY** act as party U/V.
- “VAL”—Validation Test. In the VAL test mode, The ACVP server **MUST** generate a complete (from both party U and party V’s perspectives) key agreement, and expects the IUT to be able to determine if that agreement is valid. Various types of errors **MUST** be introduced in varying portions of the key agreement process that the IUT **MUST** be able to detect and report on.

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [SP 800-56C Rev. 1] and [SP 800-56C Rev. 2].

3.2.1. Requirements Covered

- SP 800-56C—4 One-Step Key Derivation. All functionality described in the specification is covered by ACVP testing.

3.2.2. Requirements Not Covered

- SP 800-56Ar3 / SP 800-56Br2—ASN.1 encoding for the KDA is not currently supported.

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 KDA-OneStep SP800-56C 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 KDA-OneStep SP800-56C mode capabilities are advertised as JSON objects within a root “algorithm” object.

A registration **SHALL** use these properties:

Table 2 — Registration Properties

JSON Value	Description	JSON Type	Valid Values
algorithm	The algorithm under test	value	KDA
mode	The mode under test	value	OneStep
revision	The algorithm testing revision to use.	value	“Sp800-56Cr1” or “Sp800-56Cr2”
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See Section 4.2.1
auxFunctions	The auxiliary capabilities of the implementation.	array of Section 4.2.2	See Section 4.2.2
fixedInfoPattern	The pattern used for fixedInfo construction.	string	See Section 4.2.3
encoding	The encoding type to use with fixedInfo construction. Note concatenation is currently supported. ASN.1 should be coming.	array of string	concatenation
z	The domain of values representing the min/max lengths of Z the implementation can support.	Domain	
l	The length (in bits) of the largest derived keying material the implementation can produce (up to a max of 2048).	number	

4.2.1. Prerequisite Algorithms for KDA Validations

Some algorithm implementations rely on other cryptographic primitives. For example, IKEv2 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 3 — Prerequisite Algorithms

JSON Value	Description	JSON Type	Valid Values
algorithm	a prerequisite algorithm	value	DRBG, HMAC, KMAC, SHA
valValue	algorithm validation number	value	actual number or “same”
prereqAlgVal	prerequisite algorithm validation	object with algorithm and valValue properties	see above

4.2.2. AuxFunction options

Table 4 — AuxFunction Options

JSON Value	Description	JSON Type	Valid Values
auxFunctionName	The auxiliary function to use.	string	SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256, SHA3-224, SHA3-256, SHA3-384, SHA3-512, 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-

JSON Value	Description	JSON Type	Valid Values
			SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512, KMAC-128, KMAC-256
macSaltMethods	How the salt is determined (default being all 00s, random being a random salt). Required for MAC based auxFunctions.	array of string	default, random

4.2.3. FixedInfoPatternConstruction

IUTs **MUST** be capable of specifying how the FixedInfo is constructed for the KDA construction. Note that for the purposes of testing against the ACVP system, both uPartyInfo and vPartyInfo are **REQUIRED** to be registered within the fixed info pattern.

Pattern candidates:

- literal[0123456789ABCDEF]
    - uses the specified hex within "[]". literal[0123456789ABCDEF] substitutes "0123456789ABCDEF" in place of the field
- uPartyInfo
    - uPartyId { || ephemeralKey } { || ephemeralNonce } { || dkmNonce } { || c }
        - "optional" items such as ephemeralKey **MUST** be included when available for ACVP testing.
- vPartyInfo
    - vPartyId { || ephemeralKey } { || ephemeralNonce } { || dkmNonce } { || c }
        - "optional" items such as ephemeralKey **MUST** be included when available for ACVP testing.
- context
    - Random value chosen by ACVP server to represent the context.
- algorithmId
    - Random value chosen by ACVP server to represent the algorithmId.
- label
    - Random value chosen by ACVP server to represent the label.

- l
  - The length of the derived keying material in bits, **MUST** be represented in 32 bits for ACVP testing.
- t
  - A random value used to represent a secondary shared secret. Only applicable to [SP 800-56C Rev. 2].

Example (Note that party U is the server in this case "434156536964", party V is the IUT "a1b2c3d4e5"):

- "concatenation" : "literal[123456789CAFECAFE]||uPartyInfo||vPartyInfo"

Evaluated as:

- "123456789CAFECAFE434156536964a1b2c3d4e5"

4.3. Registration Example

Note there is no difference in registration properties for OneStep testing between <<SP800-56Cr1>> and <<SP800-56Cr2>> aside from the new "t" secondary shared secret that can be used for a <<SP800-56Cr2>> Fixed info pattern (See <<fixedinfopatcon>>).

Figure 2 — Registration JSON Example


```
{
  "algorithm": "KDA",
  "mode": "OneStep",
  "revision": "Sp800-56Cr1",
  "prereqVals": [
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    },
    {
      "algorithm": "SHA",
      "valValue": "123456"
    },
    {
      "algorithm": "KMAC",
      "valValue": "123456"
    },
    {
      "algorithm": "HMAC",
      "valValue": "123456"
    }
  ],
  "auxFunctions": [
    {
```

```
      "auxFunctionName": "KMAC-128",
      "macSaltMethods": [
        "default"
      ]
    }
  ],
  "fixedInfoPattern": "algorithmId||l||uPartyInfo||vPartyInfo",
  "encoding": [
    "concatenation"
  ],
  "z": [{"min": 224, "max": 8192, "increment": 8}],
  "l": 2048
}
```


Figure 3

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 KDA-OneStep SP800-56C 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 4

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 KDA-OneStep SP800-56C JSON elements of the Test Group JSON object

Table 6 — Test Group Properties

JSON Values	Description	JSON Type
tgId	Test group identifier	integer
testType	Describes the operation the client should perform on the tests data	string
tests	Array of individual test cases	See Section 5.2
kdfConfiguration	Describes the KDA configuration values used for the group	See Section 5.1.1

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.

5.1.1. KDA Configuration JSON Schema

Describes the KDA configuration for use under the test group.

Table 7 — KdfConfiguration JSON Object

JSON Value	Description	JSON Type
kdfType	The type of KDA to use for the group.	value — onestep
saltMethod	The strategy used for salting.	value — default (all 00s), random
fixedInfoPattern	The pattern used for constructing the fixedInfo.	value — See Section 4.2.3.
fixedInfoEncoding	The pattern used for constructing the fixedInfo.	value — See Section 4.2.3.
auxFunction	The auxiliary function used in the KDA.	value — See Section 4.2.2.
l	the bit length of keying material to derive from the KDA	value

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 KAS/KTS ECC test vector.

Table 8 — Test Case JSON Object

JSON Value	Description	JSON Type
tcId	Numeric identifier for the test case, unique across the entire vector set.	kdfParameter

JSON Value	Description	JSON Type
Object representing inputs into the KDA	See Section 5.2.1.	fixedInfoPartyU
Fixed information specific to party U	See Section 5.2.2.	fixedInfoPartyV

5.2.1. KDA Parameter JSON Schema

KDA specific options used for the test case.

Figure 5

Table 9 — KDF Parameter JSON Object

JSON Value	Description	JSON Type
kdfType	The type of KDA utilized.	value
salt	The salt used for the test case.	value
iv	The iv used for the test case.	value
algorithmId	The random "algorithID" used for the test case when applicable to the fixedInfo pattern.	value
context	The random "context" used for the test case when applicable to the fixedInfo pattern.	value
label	The random "label" used for the test case when applicable to the fixedInfo pattern.	value
z	shared secret z value to be used for the test case.	value
l	the bit length of keying material to derive from the KDA	value

5.2.2. FixedInfo PartyU/V JSON Schema

Fixed information that is included for party U/V for fixed info construction

Table 10 — Fixed Info JSON Object

JSON Value	Description	JSON Type
partyId	The party identifier	value
ephemeralData	Ephemeral data (randomly) included as a part of the parties fixed info construction	value

5.3. Example Test Vectors JSON

The following is a example JSON object for KDA OneStep test vectors sent from the ACVP server to the crypto module. Note there is no difference in the vector set properties for OneStep testing between [SP 800-56C Rev. 1] and [SP 800-56C Rev. 2] aside from the new “t” secondary shared secret that can be used for a [SP 800-56C Rev. 2] Fixed info pattern (See Section 4.2.3).


```
{
  "vsId": 0,
  "algorithm": "KDA",
  "mode": "OneStep",
  "revision": "Sp800-56Cr1",
  "isSample": true,
  "testGroups": [
    {
      "tgId": 1,
      "testType": "AFT",
      "tests": [
        {
          "tcId": 1,
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "z":
 "06477FB9ADB516D1D90FE0C24C1147861F5D532C2A7FEB436F83446F05765D"
          },
          "fixedInfoPartyU": {
            "partyId": "1C5B2F8FB3A411302BF1E39BB4414FD3",
            "ephemeralData":
 "A1495C3CB11BE555A283A911DD5778A3C3B5D68E59339D4EE8C256856C9918"
          },
          "fixedInfoPartyV": {
            "partyId": "5A9D98AFFD66598D7C81DBE6A13399BB",
            "ephemeralData":
 "BDF38642B74FDAF7C8DD14A2F59E9253F1AB113A9F16F694D00A7ED81710E0"
          }
        },
        {
          "tcId": 2,
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "z":
 "6C96B7341119A41000E1801FDB2D2A664F37A35C449634ECDC9A96853001DB"
          },
          "fixedInfoPartyU": {
            "partyId": "CD85C097D9BD8AC639D243ABB6649C74",
```

```
          "ephemeralData":
"4F6908C29515F35722A4803EBFD0299E35A8897DE12000F91E254D3C4B1C1D"
        },
        "fixedInfoPartyV": {
          "partyId": "3CBCF123AABD4677262F4A9B16CC4B1F",
          "ephemeralData":
"66FFF9C3F848AAFA539991BD31C1D097F05C8F3848F3F979C48FC5EF6D3B88"
        }
      },
      {
        "tcId": 3,
        "kdfParameter": {
          "kdfType": "oneStep",
          "salt": "00000000000000000000000000000000",
          "z":
"36562B4EFCE532C47AC991445E0A838103ED54471CC7E7D9F8BCDBA6C0734D"
        },
        "fixedInfoPartyU": {
          "partyId": "0025B6C42E4EA0C46F3F6A67849DCAFF",
          "ephemeralData":
"0EB59A43A4999B43179B69B67C1D9090ADAABFF673FBF98418F01A74E892EB"
        },
        "fixedInfoPartyV": {
          "partyId": "0578CC6E6E0084DAE5C26D996393E50D"
        }
      },
      {
        "tcId": 4,
        "kdfParameter": {
          "kdfType": "oneStep",
          "salt": "00000000000000000000000000000000",
          "z":
"6A2B2B0504AF9B54D6AD3C0169C10FDDDCAA7BD982320DD3A4607EE1DD36DC"
        },
        "fixedInfoPartyU": {
          "partyId": "3B6D427A4DAA7A9BC2422BFBDB21BA06",
          "ephemeralData":
"125C5F9287CDFFCCB2E0CE591E7CC8E1A79F5CCF45885F2E1478C89BA58B6B"
        },
        "fixedInfoPartyV": {
          "partyId": "3FF2FCA29CAC0D413DCE869F9982742E"
        }
      },
      {
        "tcId": 5,
        "kdfParameter": {
```

```
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "z":
"77A82D6931A138CE2314462B1166ED43E9D54C80D9A57A4FFF49A998349DA0"
          },
          "fixedInfoPartyU": {
            "partyId": "8E64E8C81E14939B637581FDA3AE9422",
            "ephemeralData":
"E55AF53197750278C1EC329EEBB62ED12296170760667A8BB0DA6765802A16"
          },
          "fixedInfoPartyV": {
            "partyId": "835C151B8217FEBABB112802528213D3",
            "ephemeralData":
"63A6691AA031FB1DA91F4395F38D801F957E42F16308692A7DEF7931994CF2"
          }
        }
      ],
      "kdfConfiguration": {
        "kdfType": "oneStep",
        "saltMethod": "default",
        "fixedInfoPattern": "uPartyInfo||vPartyInfo||l",
        "fixedInfoEncoding": "concatenation",
        "auxFunction": "KMAC-128"
      }
    },
    {
      "tgId": 62,
      "testType": "VAL",
      "tests": [
        {
          "tcId": 306,
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "150DE797DE69C94A9A539C7A3E8329FF",
            "z":
"50B69546925466939D6D69B474BF8A11299099E1D24EE00B234F8E82BAEF43429E8C5DBE"
          },
          "fixedInfoPartyU": {
            "partyId": "FEFC967B698F2F4DEBBCC93ABBAD2E7B"
          },
          "fixedInfoPartyV": {
            "partyId": "2A76DCBC3A1B2751D6D8E9189E3303E3",
            "ephemeralData":
"4A3F7D3E195290B0A0A1E8E5D44463B28291623B5669FC2103DA8427D4CE48EED4EBB256"
          },
```

```
          "dkm":
"E84EE084A05BF2B6A2A3ED8A33E44CFEB270A53F87BEF4F5447F190A15EC9A2E8706CDC2AFE0AA47460E2035
        },
        {
          "tcId": 307,
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "C81EE36994067D90AC11850A2E904B35",
            "z":
"98AE8419CEDD3CBEABAC947E4AD281235586A16AD9463A638296A03962FCC0C99A24FDD2"
          },
          "fixedInfoPartyU": {
            "partyId": "C8292D3655488EEE69F52E4F74A98A6E",
            "ephemeralData":
"DF507114129074574BEF5B698CF50611F8DF6AAD8CDE69C8C3FCA45114341BD92A50A5C8"
          },
          "fixedInfoPartyV": {
            "partyId": "C288C96A2526A5F889FD60756E946E5D"
          },
          "dkm":
"8FDF4DD962170D16523B8E6EFC7D37C4196306C1533CA95AF17C89F7A922AB8FDFBB7AFE1D50293EBEC61176
        },
        {
          "tcId": 308,
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "74D2F0F804640E0D9F67C9DAAC68744E",
            "z":
"DF950F3038B3697473BD6488FCB0B17B202E26D4777ED77C7A241CFD6CB824FEA59EF88F"
          },
          "fixedInfoPartyU": {
            "partyId": "D22FEC8F3622EDA5BEAEFBBC299CC4D9"
          },
          "fixedInfoPartyV": {
            "partyId": "416D49EC6993F74567991CA148DEF9B2"
          },
          "dkm":
"99DDEA165FB6E56B19DDDD1C549038C8420739326CCA65B674A677A26B6AC1F0CB79F107C31C9089F09F9638
        },
        {
          "tcId": 309,
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "8992D0472722054FDB68E2E7F874A9F9",
            "z":
"BC8FD6C54393D785E513BA1701F967BE0961B53852EB4C6AB4FF2B509DFA9BB100C9E0DF"
```

```
          },
          "fixedInfoPartyU": {
            "partyId": "91EFD01A0271B4E3BBA8D7AF5D237458",
            "ephemeralData":
"0B3CD301E7B5CB1149621D3DBFE7590392C3F92FBD93EEBC9779171487F6E9C8E777CB89"
          },
          "fixedInfoPartyV": {
            "partyId": "D6EB44DDC627232EBF9AD52C8A90D517",
            "ephemeralData":
"6702BBDFE0CC6C9CF6199027BE2D65386959A687B2E89691614976AC87AAAAB7E42F6CFF"
          },
          "dkm":
"067F1BDB2CB6FC773169FC7D5B049DC534B2E78C2D5AC0EFDD88CC0B58355052B3A5367D18C829D3DE2BD899
        },
        {
          "tcId": 310,
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "DBFC5DAD60398D25DD0D0C7793458BCB",
            "z":
"030B3F6CBD884428859BD0CEA63A07F896EB03B96682E72F3FE124B2D25E6AF149E7DC13"
          },
          "fixedInfoPartyU": {
            "partyId": "BC42C34BA8923F797AB18B0CC3C36906",
            "ephemeralData":
"E54E1EC091ABCCB8EFA82C1463C022AF2EB46FD49ED5B18A7F0E4B68A89C69EB2C3F8FCD"
          },
          "fixedInfoPartyV": {
            "partyId": "53760ABD8EEFFBA340A41A26FF1A409F",
            "ephemeralData":
"E432035756AE32193D18C07F7E9508B45494FEEF626CBE7C9F9BF0ADC1BE18DCCFAB3A8E"
          },
          "dkm":
"7C4E1AA4683AC60BB5938B50C3D3D164E7B7EA344F5ACFD28DC22419EA542D982E7E4176DC6F66914C9D49D3
        }
      ],
      "kdfConfiguration": {
        "kdfType": "oneStep",
        "saltMethod": "random",
        "fixedInfoPattern": "uPartyInfo||vPartyInfo||l",
        "fixedInfoEncoding": "concatenation",
        "auxFunction": "KMAC-128"
      }
    }
  ]
```

}

Figure 6 — Vector Set JSON Example

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 11 — 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

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 12 — Test Group Response Properties

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

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 object that represents a test case response for a KDA-OneStep SP800-56C.

Table 13 — Test Case Response Properties

JSON Property	Description	JSON Type
tcId	The test case identifier	integer
testPassed	Was the provided `dkm` valid? Only valid for the “VAL” test type.	boolean
dkm	The derived keying material. Provided by the IUT for “AFT” test type test cases. For single expansion tests.	hex
dkms	The derived keying materials. Provided by the IUT for “AFT” test type test cases. For multi expansion groups.	array of hex

Here is an abbreviated example of the response. Note there is no difference in the vector set response for OneStep testing between [SP 800-56C Rev. 1] and [SP 800-56C Rev. 2].

6.1. Example Test Vectors Response JSON


```
{
  "vsId": 0,
  "algorithm": "KDA",
  "mode": "OneStep",
  "revision": "Sp800-56Cr1",
  "isSample": true,
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "dkm":
 "C91D23ED66A9A9E487FC5608B26F4B66401BCC38A09F52A3DC638D73AFE474CCB70A461CFC68FDBE0C41C406
        },
        {
          "tcId": 2,
          "dkm":
 "D2DDA9897416B2994EA3C42AC36D0BD18E65A16A8DDA689EB2D1DC6D8830D902BB9D5B4BF9DE247A02E8A6AE
        },
        {
          "tcId": 3,
          "dkm":
 "7C6F142D087D055FEFEB044AF5069E0A12589039FBFEEE2206439B86452EF5591469DD82AB83E8527FA2CFD6
        },
        {
          "tcId": 4,
          "dkm":
 "38702289323D6123CB1F274106DA6F96E9CF6ACBFDEF5A17EA8A7FF07D1019B006E7FBDB454D63E7AF121F13
        },
        {
          "tcId": 5,
          "dkm":
 "63179FFD195A08CD0884D0E7556BDE51209686655267CDFADE2C7DB0462246CE2E8305F9B0AC1EA1296B21E9
        }
      ]
    },
    {
      "tgId": 62,
```

```
      "tests": [
        {
          "tcId": 306,
          "testPassed": true
        },
        {
          "tcId": 307,
          "testPassed": true
        },
        {
          "tcId": 308,
          "testPassed": true
        },
        {
          "tcId": 309,
          "testPassed": true
        },
        {
          "tcId": 310,
          "testPassed": false
        }
      ]
    }
  ]
}
```


Figure 7 — Example Response JSON SP800-56Cr1

7. Security Considerations

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

Appendix A — Revision History

Table A-1

Version	Release Date	Updates
1	2020-12-11	Initial Release

Appendix B — 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.

Elaine B. Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, Richard Davis (April 2018) *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography* (Gaithersburg, MD), April 2018. SP 800-56A Rev. 3. https://doi.org/10.6028/NIST.SP.800-56Ar3.

Elaine B. Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, Richard Davis, Scott Simon (March 2019) *Recommendation for Pair-Wise Key-Establishment Using Integer Factorization Cryptography* (Gaithersburg, MD), March 2019. SP 800-56B Rev. 2. https://doi.org/10.6028/NIST.SP.800-56Br2.

Elaine B. Barker, Lily Chen, Richard Davis (April 2018) *Recommendation for Key-Derivation Methods in Key-Establishment Schemes* (Gaithersburg, MD), April 2018. SP 800-56C Rev. 1. https://doi.org/10.6028/NIST.SP.800-56Cr1.

Elaine B. Barker, Lily Chen, Richard Davis (August 2020) *Recommendation for Key-Derivation Methods in Key-Establishment Schemes* (Gaithersburg, MD), August 2020. SP 800-56C Rev. 2. https://doi.org/10.6028/NIST.SP.800-56Cr2.

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.