ACVP KAS SSC FFC Specification

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

June 26, 2020

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

Abstract

This document defines the JSON schema for testing KAS-SSC-FFC SP800-56Ar3 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 KAS-SSC-FFC SP800-56Ar3 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 KAS-SSC-FFC SP800-56Ar3 implementations using ACVP.

2. Supported KAS SSC FFC

The following KAS algorithms **MAY** be advertised by this ACVP compliant crypto module:

- KAS-FFC-SSC / null / Sp800-56Ar3

Other KAS algorithms **MAY** be advertised by the ACVP module elsewhere.

3. Test Types and Test Coverage

This section describes the design of the tests used to validate KAS-SSC-FFC SP800-56Ar3 implementations.

3.1. Test Types

- "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 compute a shared secret `z`; 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) shared secret `z`, and expects the IUT to be able to determine if that shared secret is valid. Various types of conditions/errors MUST be introduced in varying portions of the key agreement process (changed Z, Z with leading zero nibble, etc), 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 SP800-56Ar3 KAS Shared Secret Computation (SSC).

3.2.1. Requirements Covered

- SP 800-56Ar3 — 4.1 Key Establishment Preparations. The ACVP server is responsible for generating domain parameters as per the IUT's capability registration.
- SP 800-56Ar3 — 4.2 Key-Agreement Process. Both the ACVP server and IUT participate in the shared secret computation. The server and IUT can both take the roles of party U/V, and as such the "performer" of steps depicted in "Figure 2: Key Agreement process" can vary.
- SP 800-56Ar3 — 5.6 Domain Parameters. Domain Parameter Generation SHALL be performed solely from the ACVP server, with constraints from the IUTs capabilities registration. The same set of domain parameters SHALL generate all keypairs (party U/V, static/ephemeral) for a single test case.
- SP 800-56Ar3 — 5.6 Key-Pair Generation. While Key-Pairs are used in each KAS scheme, the generation of said key-pairs is out of scope for KAS testing.
- SP 800-56Ar3 — 4.3 DLC-based Key-Transport Process / 5.7 DLC Primitives. Depending on the scheme used, either Diffie Hellman or MQV SHALL be used to negotiate a shared secret of z. Testing and validation of such key exchanges is covered under their respective schemes.
- SP 800-56Ar3 — 6 Key Agreement Schemes. All schemes specified in referenced document are supported for validation with the ACVP server.

3.2.2. Requirements Not Covered

- KAS SSC testing only covers testing of SP800-56Ar3 through the computation of a shared secret z. Additional functions of KAS as a whole such as KDF, KC, etc. MAY be covered within the scope of the full KAS testing; please see that document for further details.

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 KAS-SSC-FFC SP800-56Ar3 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. Prerequisite Algorithms for KAS FFC 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 2 — Prerequisite Algorithms JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
algorithm	a prerequisite algorithm	value	DRBG, DSA, SafePrimes, SHA	No
valValue	algorithm validation number	value	actual number or “same”	No
prereqAlgVal	prerequistie algorithm validation	object with algorithm and valValue properties	see above	Yes

KAS has conditional prerequisite algorithms, depending on the capabilities registered:

Table 3 — Prerequisite requirement conditions

Prerequisite Algorithm	Condition
DRBG	Always **REQUIRED**
SHA	Always **REQUIRED**
DSA	DSA KeyGen validation **REQUIRED** when IUT makes use of the “FB” or “FB” (legacy) domain parameters for the generation/validation of keys within the module boundary.
SafePrimes	SafePrimes KeyGen/KeyVer validation **REQUIRED** when IUT makes use of the safe-prime groups for the generation/ validation of keys within the module boundary.

4.3. Property Registration

The KAS-SSC-FFC SP800-56Ar3 mode capabilities are advertised as JSON objects within a root “algorithm” object.

A registration **SHALL** use these properties:

Table 4 — Registration Properties

JSON Property	Description	JSON Type	Valid Values
algorithm	Name of the algorithm to be validated	string	"KAS-FFC-SSC"
revision	ACVP Test version	string	"Sp800-56Ar3"
prereqVals	Prerequisites of the algorithm	object	See Section 4.1
scheme	Array of schemes supported	array of objects	See Section 4.3.1
domainParameterGenerationMethods	Array of strings representing the supported means of generating domain parameters	array of strings	See Section 4.3.2
scheme	Array of schemes supported	dictionary	See Section 4.3.1
hashFunctionZ	Optional hash function to apply to the shared secret z in instances where the IUT is unable to return the z in the clear.	string	See Section 4.3.3

4.3.1. Schemes

Schemes **MUST** be registered as a dictionary (key/value pairs) where the key is a valid Section 4.3.1.1, and the value is a Section 4.3.1.2.

4.3.1.1. Scheme Identifier

- dhEphem
- mqv1
- dhHybrid1
- mqv2
- dhHybridOneFlow

- dhOneFlow
- dhStatic

4.3.1.2. Scheme Object

The scheme object is made up of the following properties:

Table 5 — Scheme Properties

JSON Property	Description	JSON Type	Valid Values
kasRole	The roles the IUT can support for the scheme	array of string	“initiator”, “responder”

4.3.2. Domain Parameter Generation Methods

- MODP-2048
- MODP-3072
- MODP-4096
- MODP-6144
- MODP-8192
- ffdhe2048
- ffdhe3072
- ffdhe4096
- ffdhe6144
- ffdhe8192
- FB
- FC

4.3.3. Hash Function Z

An optional hash function that should be applied to `z` from both the ACVP server and IUT for comparison purposes. The provided `hashFunctionZ’s security strength MUST be at least as strong as the greatest security strength domain parameter selected from Section 4.3.2

The following hash functions **MAY** be advertised by an ACVP compliant server:

- SHA2-224
- SHA2-256
- SHA2-384
- SHA2-512
- SHA2-512/224
- SHA2-512/256

- SHA3-224
- SHA3-256
- SHA3-384
- SHA3-512

4.4. Registration Example

```
{
  "algorithm": "KAS-FFC-SSC",
  "revision": "Sp800-56Ar3",
  "scheme": {
    "dhEphem": {
      "kasRole": [
        "initiator",
        "responder"
      ]
    },
    "mqv1": {
      "kasRole": [
        "initiator"
      ]
    }
  },
  "domainParameterGenerationMethods": [
    "ffdhe2048",
    "FB"
  ],
  "hashFunctionZ": "SHA3-512"
}
```

Figure 2 — Registration JSON Example

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 KAS-SSC-FFC SP800-56Ar3 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 6 — 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 KAS-SSC-FFC SP800-56Ar3 JSON elements of the Test Group JSON object

Table 7 — 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
domainParameterGenerationMode	The curve in use for key generation for the test group	string
p	The P value for use in DSA domain parameters. Only provided for “FB”/”FC” `domainParameterGenerationMode`s	hex
q	The Q value for use in DSA domain parameters. Only provided for “FB”/”FC” `domainParameterGenerationMode`s	hex
g	The G value for use in DSA domain parameters. Only provided for “FB”/”FC” `domainParameterGenerationMode`s	hex
scheme	KAS scheme under test	string
kasRole	The IUT role under test	string
hashFunctionZ	The hash function applied to z (optional)	string
tests	Array of individual test cases	See Section 5.2

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 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 KAS-SSC-FFC SP800-56Ar3 test vector.

Table 8 — Test Case Properties

JSON Values	Description	JSON Type
tcId	Test case identifier	integer
staticPublicServer	The server static public key X. Optional depending on scheme and test type.	hex
ephemeralPublicServer	The server ephemeral public key X. Optional depending on scheme and test type.	hex

JSON Values	Description	JSON Type
staticPrivateIut	The IUT static private key. Optional depending on scheme and test type.	hex
staticPublicIut	The IUT static public key X. Optional depending on scheme and test type.	hex
ephemeralPrivateIut	The IUT ephemeral private key. Optional depending on scheme and test type.	hex
ephemeralPublicIut	The IUT ephemeral public key X. Optional depending on scheme and test type.	hex
z	The computed shared secret. Included for VAL tests when registered without a `hashFunctionZ`	hex
hashZ	The hashed computed shared secret. Included for VAL tests when registered with a `hashFunctionZ`	hex

Here is an abbreviated yet fully constructed example of the prompt


```
{
  "vsId": 1,
  "algorithm": "KAS-FFC-SSC",
  "revision": "Sp800-56Ar3",
  "testGroups": [
    {
      "p":
"A80011B59FFE93166DBA4BDA5C0E8C7C1087F101FA80CB3F8A76B018AEB56CD43D7EF9AA292F106567F8AA3A
      "q": "F36D179E1A717CA3CC6DCF504FD1C5ACAEEC65BA49E6DBD188D318D3",
      "g":
"7F7FB55DF940391E5E4E52645674126242F63F38BE1F608E3D5C9C2A9ADE5397ECA485374695D10952B9E5B7
      "tgId": 1,
      "testType": "AFT",
      "tests": [
        {
          "ephemeralPublicServer":
"890654F1CA3707EDB11EC9BE2B5917A3194C113171C0EF17D326A39E0F7B64A4E4CC8E8B423B821036883D7D
          "tcId": 1
        },
        {
```

```
        "ephemeralPublicServer":
"2BA97B4E33E3984F85FBAEEDE77487DD6AB82BC4D36E8A60095AFDDDBF7F46F0AAB810E30DB56FB559FC8A71
        "tcId": 2
      }
    ],
    "domainParameterGenerationMode": "FB",
    "scheme": "dhEphem",
    "kasRole": "initiator"
  },
  {
    "p":
"D503E21979B35BE248A73FE8EF0696DB93F51B84E00170054A9816B4C983BD9FB9E2B855E0BE1A0FD79D2B89
    "q": "9EA4AC00472C3D84984317A6629BFB80D0F359CFDFC772F740E98A47",
    "g":
"0A61782F2567CE9AD0230D5250B775C425977977A05426D584DC8D2D9F2BDEC6262C2CA7C9F0136BBE1B168A
    "tgId": 2,
    "testType": "VAL",
    "tests": [
      {
        "ephemeralPublicServer":
"8663EABE8ACAC1B376BBFA04FF4FD2080A24869584ECA51861B98B98107414411BB7CAC4E3699428E35E09EF
        "ephemeralPrivateIut":
"7FB798FD7AEED6AABA760C7C35C659D1B7B98B7480F7C4828332AA4A",
        "ephemeralPublicIut":
"42A933E40FC6E5A8BB891DAFB96A7E552147E73EBA5AF852FD5817D6F457402DDA5E524B452773EE78646504
        "tcId": 6,
        "z":
"5A226C056D2AC6F3168CB8C58FBD5924D1984E2D6FD89D63F91AFC52F6B641A70582CF1A9D9C07317E00DDAE
      },
      {
        "ephemeralPublicServer":
"2CB40CA3E239BAA9EF96EE11421018368D8D5E0EF20936795E8E423883F2160C510345D12935018FC8A316BF
        "ephemeralPrivateIut":
"5FAD24D04FE47D4050488B7714B2C8134708F828848500D131CC7254",
        "ephemeralPublicIut":
"C5D66DEBE4BAA0B88D46D20042D7B01B0E8E1C6C93D3B53799655752203182EEBDD3C66BE97E9AA1E87A0EE5
        "tcId": 7,
        "z":
"7B439FAE2BCBBCF656806252B530A0AB9FA5B37EB2924E2E88DA4B771B1E7F22273E28DA737C83B0ACD2CA9B
      }
    ],
    "domainParameterGenerationMode": "FB",
    "scheme": "dhEphem",
    "kasRole": "initiator"
  },
  {
```

```
      "tgId": 3,
      "testType": "AFT",
      "tests": [
        {
          "ephemeralPublicServer":
"489972968DA01B3357DB63F9C43F5256163546EC92F00F1EBC9E264EF8BF62B8C737104E7EEFBBDF2451CAD5
          "tcId": 11
        },
        {
          "ephemeralPublicServer":
"9741EF069DD1CAAFDED2FC7CD33CBA6C6A7667F152BE3E79DB4A3CEE64C0574E123415B62C72FAB437091371
          "tcId": 12
        }
      ],
      "domainParameterGenerationMode": "ffdhe2048",
      "scheme": "dhEphem",
      "kasRole": "initiator"
    },
    {
      "tgId": 4,
      "testType": "VAL",
      "tests": [
        {
          "ephemeralPublicServer":
"C64322705A778017EFB7BE65B1FD5CAB5CEEE0901416C8FF04A5B9D84260AB791DA6C709051F63691140514D
          "ephemeralPrivateIut":
"16B8615FDAF01C806270398C14395C1D35573B59C1D643CE7E181071584624C798421740A1E7D0DC0D0BD25F
          "ephemeralPublicIut":
"1F8A9865DCE44C1588BF8E9205A678B7BD712D1B79585F8E142704A93B5E582E994CF629BE3B6D234E99E4A5
          "tcId": 16,
          "z":
"401776D0793F06C8EBA44DCBBBB756CE8D0B3255B146ADDAD21512B0FBE50274BF664432F4E4CCF646F4FC9F
        },
        {
          "ephemeralPublicServer":
"78E94C32451BF295FA7BF02969D1915DCA629605C5255268495E063A3C179F22D35FBD3E6446F9823FCFA092
          "ephemeralPrivateIut":
"3A4740F7BC9A5A1B58BBA22EB967F9D9D9095181AF13F924AC2331552A42C05884F3B70F9BD86CEBD9BF877F
          "ephemeralPublicIut":
"2E810954FB5F3FB50943A0F1702908F5B6939E5F4A7E050D1B8A968B56E7A4603A75D449CE2885F4D7C0389E
          "tcId": 17,
          "z":
"0D2B88ED318E1FC4991D362E1ABFD6EBC4A842D722606CB6AA8294A037F820C645F1A3670DC4AB33FE5A675F
        }
      ],
      "domainParameterGenerationMode": "ffdhe2048",
```

```
      "scheme": "dhEphem",
      "kasRole": "initiator"
    }
  ]
}
```

Figure 4 — Vector Set JSON Example

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 9 — 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 10 — Test Group Response Properties

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

The following table describes the JSON object that represents a test case response for a KAS-SSC-FFC SP800-56Ar3.

Table 11 — Test Case Response Properties

JSON Property	Description	JSON Type
tcId	The test case identifier	integer
testPassed	Was the provided `z` or `hashZ` valid? Only valid for the "VAL" test type.	boolean
staticPublicIut	The IUT static public key X. Optional depending on scheme and test type.	hex
ephemeralPublicIut	The IUT ephemeral public key X. Optional depending on scheme and test type.	hex
z	The shared secret `z`. Only included for "AFT" test type when a `hashFunctionZ`	hex

JSON Property	Description	JSON Type
	is not in use.	
hashZ	The shared secret `z` run through a hash function. Only included for "AFT" test type when a `hashFunctionZ` is in use.	hex

Here is an abbreviated example of the response


```
{
  "vsId": 1,
  "algorithm": "KAS-FFC-SSC",
  "revision": "Sp800-56Ar3",
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "ephemeralPublicIut":
"94624EBEC7345E8F73D22DD3C37A721F4AEA81CFA51D35BC127C11FB3235D92E9F6F16DA88A6AE3491C08E01
          "tcId": 1,
          "z":
"45FAB5A6CF1A356034C0EA15435A9A01B7EBD2DBF3B1BC98A290D97EECB57D8C5B88FD9ECB39727DC781B4A3
        },
        {
          "ephemeralPublicIut":
"15DC1F66E6F2DF18AA44E73F45374D9452BDE4B272AD6C8D8C7D7737D3CC69DAEAADFFD26F401CD55B855AEA
          "tcId": 2,
          "z":
"A7981A88C61A87FE3BF8CC540AE4D1457B0CFB5DED0CBAB473C26B57BA8C903015294317E4B35620B1625A81
        }
      ]
    },
    {
      "tgId": 2,
      "tests": [
        {
          "tcId": 6,
          "testPassed": true
        },
        {
```

```
          "tcId": 7,
          "testPassed": true
        }
      ]
    },
    {
      "tgId": 3,
      "tests": [
        {
          "ephemeralPublicIut":
"53DC079D7082909E715064A90A3A5D054C323F327B40D363FE4238BD3C4C293B81CB6E68C2A6AF65511FE0B3
          "tcId": 11,
          "z":
"1F6D92321078DEBA865431274DBF2A320CF51B92CCC630C0684830D43D6107F0477191C65D4A57D3BD9B86C2
        },
        {
          "ephemeralPublicIut":
"8E3B2EE3E566AEF03517E1925886BA3FA956EC5E8309FAD7CE302374801FB6A994620F5756AFE678F2E2DC80
          "tcId": 12,
          "z":
"F46F8D095883CE9567162469FEF1440F31CB349C90B2A0FCC57B92BBD46E377B86F3C92647A8FDB03385B8D7
        }
      ]
    },
    {
      "tgId": 4,
      "tests": [
        {
          "tcId": 16,
          "testPassed": false
        },
        {
          "tcId": 17,
          "testPassed": true
        }
      ]
    }
  ]
}
```


Figure 5 — Example Response JSON

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	2020-06-26	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.

T. Kivinen, M. Kojo (May 2003) *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)* (Internet Engineering Task Force), RFC 3526, May 2003. RFC 3526. RFC RFC3526. DOI 10.17487/RFC3526. https://www.rfc-editor.org/info/rfc3526.

D. Gillmor (August 2016) *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)* (Internet Engineering Task Force), RFC 7919, August 2016. RFC 7919. RFC RFC7919. DOI 10.17487/RFC7919. https://www.rfc-editor.org/info/rfc7919.

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 2013) *Digital Signature Standard (DSS)* (Gaithersburg, MD), July 2013. FIPS 186-4. https://doi.org/10.6028/NIST.FIPS.186-4.

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.

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.