ACVP KAS FFC SP800-56Ar3 JSON Specification

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

January 31, 2020

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

Abstract

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

2. Supported KAS-FFCs

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

- KAS-FFC / null / Sp800-56Ar3

3. Test Types and Test Coverage

The ACVP server performs a set of tests on 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 KAS capabilities. This section describes the design of the tests used to validate implementations of KAS algorithms.

3.1. Test Types

There are two test types for KAS 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, as well as recipient/provider to key confirmation.
- "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 (changed DKM, changed key, changed hash digest, 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 [SP 800-56A Rev. 3].

3.2.1. KAS-FFC Requirements Covered

- SP 800-56Ar3—5.1 Cryptographic Hash Functions. SHA1, SHA2, and SHA3 hash functions **SHALL** be available for the various pieces of KAS requiring use of a hash function; such as approved MACs and OneStep KDF.
- SP 800-56Ar3—5.2 Message Authentication Code (MAC) Algorithms. AES-CMAC, HMAC, and KMAC algorithms **SHALL** be available for testing under KDFs and KC as the specification states.
- SP 800-56Ar3—5.3 Random Number Generation. Though random values are used, the testing of the construction of those random values **SHALL NOT** be in scope of ACVP testing.
- SP 800-56Ar3—5.4 Nonces. Though nonces are used, the testing of the construction of those nonces **SHALL NOT** be in scope of ACVP testing.
- SP 800-56Ar3—5.5 Domain Parameters. Domain Parameters **SHALL** be used in the testing of KAS as per the specification, though the generation of those parameters is outside the scope of testing.
- SP 800-56Ar3—5.6 Key-Pair Generation. Each KAS scheme from one or both parties utilizes a key pair for arriving at a shared secret, and deriving a key. Though a key pair(s)

are utilized in ACVP testing, the testing of the generation of said key pairs is outside the scope of this testing.

- SP 800-56Ar3 — 5.7 DLC Primitives. Diffie Hellman and MQV **SHALL** be tested under their respective KAS schemes.
- SP 800-56Ar3 — 5.8 Key-Derivation Methods for Key-Establishment Schemes. The ACVP server **SHALL** make various KDFs available for testing. The KDFs covered under ACVP server testing **SHALL** include the KDFs specified in SP800-56B, SP800-56C, SP800-108, and SP800-135 (where applicable).
- SP 800-56Ar3 — 5.9 KeyConfirmation. The ACVP server **SHALL** support key confirmation for applicable KAS and KTS schemes.
- SP 800-56Ar3 — 6 Key Agreement Schemes. The ACVP server **SHALL** support testing for all KAS schemes specified in the SP800-56Ar3 document.
- SP 800-56Cr1 — 4 One-Step Key Derivation. One-Step Key Derivation testing **SHALL** be supported by the ACVP server. FixedInfo construction is covered within the ACVP specification, and can be tailored to the IUTs needs. ASN.1 format of fixedInfo construction (currently) is NOT supported.
- SP 800-56Cr1 — 5 Two-Step Key Derivation. Two-Step Key Derivation testing **SHALL** be supported by the ACVP server. FixedInfo construction is covered within the ACVP specification, and can be tailored to the IUTs needs. ASN.1 format of fixedInfo construction (currently) is NOT supported.
- SP 800-108 — 4 Pseudorandom Function (PRF). All iterations of the KDF described in SP800-108 use a separate PRF. All implementations of the PRF **SHALL** be available for testing through the ACVP server generated tests.
- SP 800-108 — 5 Key Derivation Functions (KDF). The three implementations of KDFs in SP800-108 **SHALL** be available for testing through the ACVP Server.

3.2.2. KAS-FFC Requirements Not Covered

- SP 800-56Ar3 — 4.3 DLC-based Key-Transport Process. KeyWrapping is not incorporated into ACVP testing.
- SP 800-56Ar3 — 5.3 Random Number Generation. Though random values are used, the testing of the construction of those random values **SHALL NOT** be in scope of ACVP testing.
- SP 800-56Ar3 — 5.4 Nonces. Though nonces are used, the testing of the construction of those nonces **SHALL NOT** be in scope of ACVP testing.
- SP 800-56Ar3 — 5.5 Domain Parameters. Domain Parameters **SHALL** be used in the testing of KAS as per the specification, though the generation of those parameters is outside the scope of testing.
- SP 800-56Ar3 — 5.6 Key-Pair Generation. Each KAS scheme from one or both parties utilizes a key pair for arriving at a shared secret, and deriving a key. Though a key pair(s) are utilized in ACVP testing, the testing of the generation of said key pairs is outside the scope of this testing.

- SP 800-56Ar3 — 5.6.2 Required Assurances. IUT assurance testing is outside the scope of ACVP testing.
- SP 800-56Ar3 — 5.6.2 Key Pair Management. Testing the IUT's management of Key Pairs is outside the scope of ACVP testing.
- SP 800-56Ar3 — 5.8.1.2 The ASN.1 Format for FixedInfo. The ACVP server (currently) **SHALL NOT** support the testing of this format of fixed info.
- SP 800-56Ar3 — 7 Rationale for Selecting a Specific Scheme. There is no testing associated with the IUT's choice of selecting a specific scheme.
- SP 800-56Ar3 — 8 Key Recovery. Key Recovery **SHALL NOT** be within the scope of ACVP testing.
- SP 800-56Cr1 — 4 One-Step Key Derivation. ASN.1 format of fixedInfo construction (currently) is NOT supported.
- SP 800-56Cr1 — 5 Two-Step Key Derivation. ASN.1 format of fixedInfo construction (currently) is NOT supported.
- SP 800-56Cr1 — 7 Selecting Hash Functions and MAC Algorithms. The process that goes into the selection of Hash functions and MAC algorithms **SHALL NOT** be in scope of ACVP testing, though the ACVP server **SHALL** support all indicated Hash and MAC functions.
- SP 800-56Cr1 — 7 Selecting Hash Functions and MAC Algorithms. The process that goes into the selection of Hash functions and MAC algorithms **SHALL NOT** be in scope of ACVP testing, though the ACVP server **SHALL** support all indicated Hash and MAC functions.

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 FFC 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	CMAC, DRBG, DSA, HMAC, KMAC, SafePrimes, SHA, SP800-108	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 "FB" or "FB" (legacy) domain parameters for the generation/validation of keys within the module boundary.
CMAC	CMAC validation **REQUIRED** when IUT is performing KeyConfirmation (KC) or a KDF and utilizing CMAC.
HMAC	HMAC validation **REQUIRED** when IUT is performing KeyConfirmation (KC) or a KDF and utilizing HMAC.
KMAC	KMAC validation **REQUIRED** when IUT is performing KeyConfirmation (KC) or a KDF and utilizing KMAC.

4.3. KAS FFC Algorithm Capabilities JSON Values

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

Table 4 — Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
algorithm	The algorithm under test	value	KAS-FFC	No
revision	The algorithm testing revision to use.	value	“Sp800-56Ar3”	No
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See Section 4.2	No
function	Type of function supported	array	See Section 4.4	Yes
iutId	The identifier of the IUT.	hex		No
scheme	Array of supported key agreement schemes each having their own capabilities	object	See Section 4.5.1	No
domainParameterGenerationMethods	Array of IUT supported domain parameter generation methods. Note that “FB” and “FC” are considered legacy, and should only be included for interoperability with other systems unable to support the safe prime groups.	array	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, FB, FC	No

Note: Some optional values are **REQUIRED** depending on the algorithm. Failure to provide these values will result in the ACVP server returning an error to the ACVP client during registration.

4.4. Supported KAS FFC Functions

The following function types **MAY** be advertised by the ACVP compliant crypto module:

- keyPairGen—IUT can perform keypair generation.
- partialVal—IUT can perform partial public key validation ([SP 800-56A Rev. 3] section 5.6.2.3)

- fullVal — IUT can perform full public key validation ([SP 800-56A Rev. 3] section 5.6.2.3)

4.5. KAS FFC Schemes

All other scheme capabilities are advertised as a self-contained JSON object using the following values. Note that AT LEAST one valid scheme must be registered.

4.5.1. KAS FFC Scheme Capabilities JSON Values

- dhHybrid1
- mqv2
- dhEphem — KeyConfirmation not supported.
- dhHybridOneFlow
- mqv1
- dhOneFlow — Can only provide unilateral key confirmation party V to party U.
- dhStatic

Table 5 — KAS FFC Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
kasRole	Roles supported for key agreement	array	initiator and/or responder	No
kdfMethods	The KDF methods to use when testing KAS schemes.	object	Section 4.5.1.1	No
keyConfirmationMethod	The KeyCnfirmation capabilities (when supported) for the scheme.	object	Section 4.5.1.2	Yes
	The length of the key to derive (using a KDF) or transport (using a KTS scheme). This value should be large enough to accommodate the key length used for the mac algorithms in use for key confirmation, ideally the maximum value the IUT can support with their KAS/KTS implementation. Maximum value (for	integer	128 minimum without KC, 136 minimum with KC, maximum 1024.	No

JSON Value	Description	JSON type	Valid Values	Optional
	testing purposes) is 1024.			

4.5.1.1. Supported Kdf Methods

Note that AT LEAST one KDF Method is required for KAS schemes. The following **MAY** be advertised by the ACVP compliant crypto module:

Table 6 — KDF Options

JSON Value	Description	JSON type	Valid Values	Optional
oneStepKdf	Indicates the IUT will be testing key derivation using the SP800-56Cr1 OneStepKdf.	object	Section 4.5.1.1.1	Yes
oneStepNoCounterKdf	Indicates the IUT will be testing key derivation using the SP800-56Cr1 OneStepNoCounterKdf.	object	Section 4.5.1.1.2	Yes
twoStepKdf	Indicates the IUT will be testing key derivation using the SP800-56Cr1 TwoStepKdf.	object	Section 4.5.1.1.3	Yes

4.5.1.1.1. One Step KDF Capabilities

Table 7 — One Step KDF Options

JSON Value	Description	JSON type	Valid Values	Optional
auxFunctions	The auxiliary functions to use with the KDF.	array of Table 8	See Table 8	No
fixedInfoPattern	The pattern used for fixedInfo construction.	string	See Section 4.5.1.3	No
encoding	The encoding type to use with fixedInfo construction. Note concatenation is currently supported. ASN.1 should be coming.	array of string	concatenation	No

Table 8 — AuxFunction Options

JSON Value	Description	JSON type	Valid Values	Optional
auxFunctionName	The auxiliary function to use. Note that a customization string of "KDF" is used for the function when KMAC is utilized.	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-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512, KMAC-128, KMAC-256	No
macSaltMethod	How the salt is determined (default being all 00s, random being a random salt).	array of string	default, random	Not optional for mac based auxiliary functions.

4.5.1.1.2. One Step No Counter KDF Capabilities

The one step no counter KDF is a special implementation of the one step KDF. This implementation of the KDF does not utilize a 32 bit counter as a part of the concatenation that gets fed into function `H`. As such, there is no loop within the KDF due to there being no information changing between iterations of the potential concatenation, and the KDF output length is capped at the output length of the chosen aux function (or 2048 in the case of KMAC).

Table 9 — One Step No Counter KDF Options

JSON Value	Description	JSON Type	Valid Values	Optional
auxFunctions	The auxiliary functions to use with the KDF.	array of Table 10	See Table 10	No
fixedInfoPattern	The pattern used for fixedInfo construction.	string	See Section 4.5.1.3	No
encoding	The encoding type to use with fixedInfo construction. Note concatenation is currently supported. ASN.1 should be coming.	array of string	concatenation	No

Table 10 — AuxFunction Options

JSON Value	Description	JSON Type	Valid Values	Optional
auxFunctionName	The auxiliary function to use. Note that a customization string of “KDF” is used for the function when KMAC is utilized.	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-SHA3-256, HMAC-	No

JSON Value	Description	JSON Type	Valid Values	Optional
			SHA3-384, HMAC-SHA3-512, KMAC-128, KMAC-256	
	The length of the keying material to derive (cannot exceed output length of aux function)	No	macSaltMethods	How the salt is determined (default being all 00s, random being a random salt).

4.5.1.1.3. Two Step KDF Capabilities

Table 11 — Two Step KDF Options

JSON Value	Description	JSON type	Valid Values	Optional
capabilities	The capabilities supported for the Two Step KDF.	array of Table 12	See Table 12	No

Note this capabilities object is very similar to the capability object from SP800-108.

Table 12 — TwoStepCapabilities Options

JSON Value	Description	JSON type	Valid Values	Optional
macSaltMethod	How the salt is determined (default being all 00s, random being a random salt).	array of string	default, random	Not optional for mac based auxiliary functions.
fixedInfoPattern	The pattern used for fixedInfo construction.	string	See Section 4.5.1.3	No
encoding	The encoding type to use with fixedInfo construction. Note concatenation is currently supported. ASN.1 should be coming.	array of string	concatenation	No
kdfMode	The strategy for running the KDF.	string	counter, fedback, double pipeline iteration	No
macMode	The macMode supported by the KDF.	array of string	CMAC-AES128, CMAC-AES192, CMAC-AES256,	No

JSON Value	Description	JSON type	Valid Values	Optional
			HMAC-SHA-1, HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA2-512/224, HMAC-SHA2-512/256, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	
fixedDataOrder	The counter locations supported by the KDF.	array of string	none, before fixed data, after fixed data, before iterator	No
counterLength	The counter lengths supported for the KDF.	array of integer	8, 16, 24, 32	Not optional for counter mode.
supportedLengths	The supported derivation lengths.	domain	Single range (of literal) expected. Registered value must support the L value provided.	No
supportsEmptyIv	The KDF supports an empty IV (feedback mode).	boolean	true, false	No
requiresEmptyIv	The KDF requires an empty IV (feedback mode).	boolean	true, false	Yes

4.5.1.2. Supported KeyConfirmation Method

Table 13 — KAS FFC KeyConfirmation Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
macMethods	The MAC methods to use when testing KAS or KTS schemes with key confirmation.	object	Section 4.5.1.4	No
keyConfirmationDirections	The directions in which key confirmation is supported.	array	unilateral, bilateral	No

JSON Value	Description	JSON type	Valid Values	Optional
keyConfirmationRoles	The roles in which key confirmation is supported.	array	provider, recipient	No

4.5.1.3. FixedInfoPatternConstruction

IUTs **MUST** be capable of specifying how the FixedInfo is constructed for the KAS/KTS negotiation. 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.

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”

4.5.1.4. Supported MAC Methods

Note that AT LEAST one mac method must be supplied when making use of Key Confirmation.

Table 14 — MAC Method Options

JSON Value	Description	JSON type	Valid Values	Optional
CMAC	Utilizes CMAC as the MAC algorithm.	object	See Section 4.5.1.4.1. Note that the keyLen must be 128, 192, or 256 for this MAC.	Yes
HMAC-SHA-1	Utilizes HMAC-SHA-1 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA2-224	Utilizes HMAC-SHA2-224 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA2-256	Utilizes HMAC-SHA2-256 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA2-384	Utilizes HMAC-SHA2-384 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA2-512	Utilizes HMAC-SHA2-512 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA2-512/224	Utilizes HMAC-SHA2-512/224 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA2-512/256	Utilizes HMAC-SHA2-512/256 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA3-224	Utilizes HMAC-SHA3-224 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA3-256	Utilizes HMAC-SHA3-256 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA3-384	Utilizes HMAC-SHA3-384 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
HMAC-SHA3-512	Utilizes HMAC-SHA3-512 as the MAC algorithm.	object	See Section 4.5.1.4.1	Yes
KMAC-128	Utilizes KMAC-128 as the MAC algorithm. Note that a customization string of “KC” is used for the function when KMAC is utilized for Key Confirmation.	object	See Section 4.5.1.4.1	Yes
KMAC-256	Utilizes KMAC-256 as the MAC algorithm. Note that a	object	See Section 4.5.1.4.1	Yes

JSON Value	Description	JSON type	Valid Values	Optional
	customization string of "KC" is used for the function when KMAC is utilized for Key Confirmation.			

4.5.1.4.1. Supported MAC Options

Table 15 — MAC Method Base Options

JSON Value	Description	JSON type	Valid Values	Optional
keyLen	The amount of bits from the DKM to pass into the KeyConfirmation MAC function.	integer	128 — 512. Note that the DKM is **Required** to have at least 8 bits available after subtracting the keyLen specified.	No
macLen	The amount of bits to use as the tag from the MAC function.	integer	64 — 512.	No

4.6. Example KAS-FFC Registration

The following is a example JSON object advertising support for KAS FFC.

```
{
  "algorithm": "KAS-FFC",
  "revision": "Sp800-56Ar3",
  "prereqVals": [
    {
      "algorithm": "DSA",
      "valValue": "123456"
    },
    {
      "algorithm": "SafePrimes",
      "valValue": "123456"
    },
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    },
    {
      "algorithm": "SHA",
      "valValue": "123456"
    },
    {
```

```
      "algorithm": "KMAC",
      "valValue": "123456"
    },
    {
      "algorithm": "HMAC",
      "valValue": "123456"
    }
  ],
  "iutId": "123456ABCD",
  "scheme": {
    "dhStatic": {
      "kasRole": [
        "initiator",
        "responder"
      ],
      "kdfMethods": {
        "oneStepKdf": {
          "auxFunctions": [
            {
              "auxFunctionName": "KMAC-128",
              "macSaltMethods": [
                "default"
              ]
            }
          ],
          "fixedInfoPattern": "algorithmId||l||uPartyInfo||vPartyInfo",
          "encoding": [
            "concatenation"
          ]
        },
        "oneStepNoCounterKdf": {
          "auxFunctions": [
            {
              "auxFunctionName": "KMAC-128",
              "l": 256,
              "macSaltMethods": [
                "default"
              ]
            }
          ],
          "fixedInfoPattern": "algorithmId||l||uPartyInfo||vPartyInfo",
          "encoding": [
            "concatenation"
          ]
        },
        "twoStepKdf": {
```

```
        "capabilities": [
          {
            "macSaltMethods": [
              "random"
            ],
            "fixedInfoPattern": "l||label||uPartyInfo||vPartyInfo||context",
            "encoding": [
              "concatenation"
            ],
            "kdfMode": "feedback",
            "macMode": [
              "HMAC-SHA3-224"
            ],
            "supportedLengths": [
              512
            ],
            "fixedDataOrder": [
              "after fixed data"
            ],
            "counterLength": [
              32
            ],
            "requiresEmptyIv": false,
            "supportsEmptyIv": false
          }
        ]
      }
    },
    "keyConfirmationMethod": {
      "macMethods": {
        "KMAC-128": {
          "keyLen": 128,
          "macLen": 128
        }
      },
      "keyConfirmationDirections": [
        "unilateral",
        "bilateral"
      ],
      "keyConfirmationRoles": [
        "provider",
        "recipient"
      ]
    },
    "l": 512
  }
```

```
  },
  "domainParameterGenerationMethods": [
    "ffdhe2048",
    "FB"
  ]
}
```

Figure 2

5. Generation Requirements per Party per Scheme

The various schemes of KAS all have their own requirements as to keys and nonces per scheme, per party. The below table demonstrates those generation requirements:

Table 16 — Required Party Generation Obligations

Scheme	KasMode	KasRole	KeyConfirmationRole	KeyConfirmationDirection	StaticKeyPair	EphemeralKeyPair	EphemeralNonce	DkmNonce
DhHybrid1	NoKdfNoKc	InitiatorParty	None	None	True	True	False	False
DhHybrid1	NoKdfNoKc	ResponderParty	None	None	True	True	False	False
DhHybrid1	KdfNoKc	InitiatorParty	None	None	True	True	False	False
DhHybrid1	KdfNoKc	ResponderParty	None	None	True	True	False	False
DhHybrid1	KdfKc	InitiatorParty	Provider	Unilateral	True	True	False	False
DhHybrid1	KdfKc	InitiatorParty	Provider	Bilateral	True	True	False	False
DhHybrid1	KdfKc	InitiatorParty	Recipient	Unilateral	True	True	False	False
DhHybrid1	KdfKc	InitiatorParty	Recipient	Bilateral	True	True	False	False
DhHybrid1	KdfKc	ResponderParty	Provider	Unilateral	True	True	False	False
DhHybrid1	KdfKc	ResponderParty	Provider	Bilateral	True	True	False	False
DhHybrid1	KdfKc	ResponderParty	Recipient	Unilateral	True	True	False	False
DhHybrid1	KdfKc	ResponderParty	Recipient	Bilateral	True	True	False	False
Mqv2	NoKdfNoKc	InitiatorParty	None	None	True	True	False	False
Mqv2	NoKdfNoKc	ResponderParty	None	None	True	True	False	False
Mqv2	KdfNoKc	InitiatorParty	None	None	True	True	False	False
Mqv2	KdfNoKc	ResponderParty	None	None	True	True	False	False
Mqv2	KdfKc	InitiatorParty	Provider	Unilateral	True	True	False	False
Mqv2	KdfKc	InitiatorParty	Provider	Bilateral	True	True	False	False
Mqv2	KdfKc	InitiatorParty	Recipient	Unilateral	True	True	False	False
Mqv2	KdfKc	InitiatorParty	Recipient	Bilateral	True	True	False	False
Mqv2	KdfKc	ResponderParty	Provider	Unilateral	True	True	False	False
Mqv2	KdfKc	ResponderParty	Provider	Bilateral	True	True	False	False
Mqv2	KdfKc	ResponderParty	Recipient	Unilateral	True	True	False	False
Mqv2	KdfKc	ResponderParty	Recipient	Bilateral	True	True	False	False
DhEphem	NoKdfNoKc	InitiatorParty	None	None	False	True	False	False
DhEphem	NoKdfNoKc	ResponderParty	None	None	False	True	False	False
DhEphem	KdfNoKc	InitiatorParty	None	None	False	True	False	False
DhEphem	KdfNoKc	ResponderParty	None	None	False	True	False	False
DhHybridOneFlow	NoKdfNoKc	InitiatorParty	None	None	True	True	False	False
DhHybridOneFlow	NoKdfNoKc	ResponderParty	None	None	True	False	False	False
DhHybridOneFlow	KdfNoKc	InitiatorParty	None	None	True	True	False	False
DhHybridOneFlow	KdfNoKc	ResponderParty	None	None	True	False	False	False
DhHybridOneFlow	KdfKc	InitiatorParty	Provider	Unilateral	True	True	False	False
DhHybridOneFlow	KdfKc	InitiatorParty	Provider	Bilateral	True	True	False	False
DhHybridOneFlow	KdfKc	InitiatorParty	Recipient	Unilateral	True	True	False	False
DhHybridOneFlow	KdfKc	InitiatorParty	Recipient	Bilateral	True	True	False	False

Scheme	KasMode	KasRole	KeyConfirmationRole	KeyConfirmationDirection	StaticKeyPair	EphemeralKeyPair	EphemeralNonce	DkmNonce
DhHybridOneFlow	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
DhHybridOneFlow	KdfKc	ResponderPartyV	Provider	Bilateral	True	False	True	False
DhHybridOneFlow	KdfKc	ResponderPartyV	Recipient	Unilateral	True	False	True	False
DhHybridOneFlow	KdfKc	ResponderPartyV	Recipient	Bilateral	True	False	True	False
Mqv1	NoKdfNoKc	InitiatorPartyU	None	None	True	True	False	False
Mqv1	NoKdfNoKc	ResponderPartyV	None	None	True	False	False	False
Mqv1	KdfNoKc	InitiatorPartyU	None	None	True	True	False	False
Mqv1	KdfNoKc	ResponderPartyV	None	None	True	False	False	False
Mqv1	KdfKc	InitiatorPartyU	Provider	Unilateral	True	True	False	False
Mqv1	KdfKc	InitiatorPartyU	Provider	Bilateral	True	True	False	False
Mqv1	KdfKc	InitiatorPartyU	Recipient	Unilateral	True	True	False	False
Mqv1	KdfKc	InitiatorPartyU	Recipient	Bilateral	True	True	False	False
Mqv1	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
Mqv1	KdfKc	ResponderPartyV	Provider	Bilateral	True	False	True	False
Mqv1	KdfKc	ResponderPartyV	Recipient	Unilateral	True	False	True	False
Mqv1	KdfKc	ResponderPartyV	Recipient	Bilateral	True	False	True	False
DhOneFlow	NoKdfNoKc	InitiatorPartyU	None	None	False	True	False	False
DhOneFlow	NoKdfNoKc	ResponderPartyV	None	None	True	False	False	False
DhOneFlow	KdfNoKc	InitiatorPartyU	None	None	False	True	False	False
DhOneFlow	KdfNoKc	ResponderPartyV	None	None	True	False	False	False
DhOneFlow	KdfKc	InitiatorPartyU	Recipient	Unilateral	False	True	False	False
DhOneFlow	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
DhStatic	NoKdfNoKc	InitiatorPartyU	None	None	True	False	False	False
DhStatic	NoKdfNoKc	ResponderPartyV	None	None	True	False	False	False
DhStatic	KdfNoKc	InitiatorPartyU	None	None	True	False	False	True
DhStatic	KdfNoKc	ResponderPartyV	None	None	True	False	False	False
DhStatic	KdfKc	InitiatorPartyU	Provider	Unilateral	True	False	False	True
DhStatic	KdfKc	InitiatorPartyU	Provider	Bilateral	True	False	False	True
DhStatic	KdfKc	InitiatorPartyU	Recipient	Unilateral	True	False	False	True
DhStatic	KdfKc	InitiatorPartyU	Recipient	Bilateral	True	False	False	True
DhStatic	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
DhStatic	KdfKc	ResponderPartyV	Provider	Bilateral	True	False	True	False
DhStatic	KdfKc	ResponderPartyV	Recipient	Unilateral	True	False	True	False
DhStatic	KdfKc	ResponderPartyV	Recipient	Bilateral	True	False	True	False

6. 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 SP800-56Ar3 KAS FFC 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 17 — 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 6.1	array

An example of this would look like this

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

Figure 3

6.1. 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 secure hash JSON elements of the Test Group JSON object.

The test group for KAS/KTS FFC is as follows:

Table 18 — Vector Group JSON Object

JSON Value	Description	JSON type	Optional
tgId	Numeric identifier for the test group, unique across the entire vector set.	value	No
testType	The type of test for the group (AFT or VAL).	value	No
scheme	The scheme in use for the group. See Section 4.5.1 for possible values.	value	No
kasRole	The group role from the perspective of the IUT.	value	No
	The length of key to derive/transport.	value	No
iutId	The Iut's identifier.	value	No
serverId	The ACVP server's identifier.	value	No
kdfConfiguration	The KDF configuration for the group.	Object, See Section 6.1.1	No
macConfiguration	The MAC configuration for the group.	Object, See Section 6.1.2	Not optional for schemes using key confirmation.
keyConfirmationDirection	The key confirmation direction.	value	Yes
keyConfirmationRole	The key confirmation role.	value	Yes
domainParameterGenerationMode	The domain parameter type used.	value	No
	The P value when using FIPS 186-* type domain parameter generation for groups using a domainParameterGenerationMode of "FB" or "FC".	value	Yes
	The Q value when using FIPS 186-* type domain parameter generation for groups using a domainParameterGenerationMode of "FB" or "FC".	value	Yes
	The G value when using FIPS 186-* type domain parameter generation for groups using a domainParameterGenerationMode of "FB" or "FC".	value	Yes
tests	The tests for the group.	Array of objects, See Section 6.2.	No

6.1.1. KDF Configuration JSON Schema

Describes the KDF configuration for use under the test group.

Table 19 — KdfConfiguration JSON Object

JSON Value	Description	JSON type	Optional
kdfType	The type of KDF to use for the group.	value — onestep, twostep	No
satMethod	The strategy used for salting.	value — default (all 00s), random	No
fixedInfoPattern	The pattern used for constructing the fixedInfo.	value — See Section 4.5.1.3.	No
fixedInfoEncoding	The pattern used for constructing the fixedInfo.	value — See Section 4.5.1.3.	No
auxFunction	The auxiliary function used in the KDF.	value — See Table 8.	Not optional for OneStepKdf.
macMode	The MAC function used in the KDF.	value — See Table 12.	Not optional for TwoStepKdf.
counterLocation	The counter location.	value	Yes
counterLen	The counter length.	value	Yes
ivLen	The iv length.	value	Yes

6.1.2. MAC Configuration JSON Schema

Describes the key confirmation MAC configuration for use under the test group.

Table 20 — MacConfiguration JSON Object

JSON Value	Description	JSON type	Optional
macType	The macType used in key confirmation.	value — HMAC-SHA2-224, HMAC-SHA2-256, HMAC-SHA2-384, HMAC-SHA2-512, HMAC-SHA2-512/224, HMAC-SHA2-512/256, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512, CMAC, KMAC-128, KMAC-256	No

JSON Value	Description	JSON type	Optional
keyLen	The number of bits to take from the DKM to use for the mac key in key confirmation.	value	No
macLen	The number of bits to use for the MAC tag.	value	No

6.2. 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 FFC test vector.

Table 21 — Test Case JSON Object

JSON Value	Description	JSON type	Optional
tcId	Numeric identifier for the test case, unique across the entire vector set.	value	No
ephemeralPublicKeyIut	The IUT's ephemeral public key.	value	Yes
staticPublicKeyIut	The IUT's static public key.	value	Yes
ephemeralPublicKeyServer	The Server's ephemeral public key.	value	Yes
staticPublicKeyServer	The Server's static public key.	value	Yes
dkmNonceIut	The IUT's nonce used in static schemes for Key Confirmation.	value	Yes
ephemeralNonceIut	The IUT's ephemeral nonce used in some schemes.	value	Yes
dkmNonceServer	The Server's nonce used in static schemes for Key Confirmation.	value	Yes
ephemeralNonceServer	The Server's ephemeral nonce used in some schemes.	value	Yes
staticPrivateKeyIut	The IUT's static private key.	value	Yes
ephemeralPrivateKeyIut	The IUT's ephemeral private key.	value	Yes

JSON Value	Description	JSON type	Optional
kdfParameter	The KDF parameters for this test case.	value — See Section 6.2.1.	Yes
dkm	The derived keying material.	value	Yes
tag	The tag generated as a part of key conformation (from the IUT perspective).	value	Yes

6.2.1. KDF Parameter JSON Schema

KDF specific options used for the test case.

Table 22 — KDF Parameter JSON Object

JSON Value	Description	JSON type	Optional
kdfType	The type of KDF utilized.	value	No
salt	The salt used for the test case.	value	Yes
iv	The iv used for the test case.	value	Yes
algorithmId	The random "algorithID" used for the test case when applicable to the fixedInfo pattern.	value	Yes
context	The random "context" used for the test case when applicable to the fixedInfo pattern.	value	Yes
label	The random "label" used for the test case when applicable to the fixedInfo pattern.	value	Yes

6.3. Example Test Vectors JSON Object KAS-FFC

The following is a example JSON object for KAS-FFC test vectors sent from the ACVP server to the crypto module.

```
{
  "vsId": 0,
  "algorithm": "KAS-FFC",
  "revision": "Sp800-56Ar3",
  "isSample": true,
  "testGroups": [
    {
      "tgId": 1,
      "testType": "AFT",
      "tests": [
```

```
        {
          "staticPublicServer":
"45AE066A16032CA624E949382043512DA74A09819DE7FA2420BAF7CDEBB003DC74198398C64DCE50A752EAF9
          "staticPublicIut":
"BD32AFAB7EBAF8DAAEAB8B057D9C31CAF177D2DBB8D2B423EA16F4FD72277728F50B1C0B2E4C4A1103502115
          "tcId": 1,
          "ephemeralNonceServer":
"2112802D475BCB471F773F66BE4219D1A3EB9A561EB7C92CEC98149CEBAAECC338F049F5C11A0198EA3A3C28
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "6DD507DEEA57DE300CF2ADDE72D8990C"
          }
        }
      ],
      "domainParameterGenerationMode": "ffdhe2048",
      "scheme": "dhStatic",
      "kasRole": "initiator",
      "l": 512,
      "iutId": "123456ABCD",
      "serverId": "434156536964",
      "kdfConfiguration": {
        "kdfType": "oneStep",
        "saltMethod": "default",
        "fixedInfoPattern": "algorithmId||l||uPartyInfo||vPartyInfo",
        "fixedInfoEncoding": "concatenation",
        "auxFunction": "KMAC-128"
      },
      "macConfiguration": {
        "macType": "KMAC-128",
        "keyLen": 128,
        "macLen": 128
      },
      "keyConfirmationDirection": "unilateral",
      "keyConfirmationRole": "provider"
    },
    {
      "p":
"9164E286EB8702861278F4A1CE88B549529B9BAADF2763417F884F0DB963336AD4BBFC28915EB17B2F44B234
      "q": "F1CEB8B3D387728D521E692D03E86D2A5F52C37ECB0A4087E3168EBF",
      "g":
"7A17F66192CA65FAA9C4510A1C419695BD57055A614D5240FE4C030CFD0E59D95317DC37EAF5F7D24C3E9612
      "tgId": 9,
      "testType": "AFT",
      "tests": [
        {
```

```
          "staticPublicServer":
"01F07267C7F628EE8ADBBEB8707222F5CC6352BA2C3F263381063D878C4A900EC7C827899E3CC544638E893D
          "tcId": 81,
          "ephemeralNonceServer":
"AEC3FD3F03A80C9806F41A1F94DCE6E29F7430DEF1A2CE680250D0FB321E4625459A5D57FE235847A250AA71
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "001E83EA55F912BAE17F76291298C3B8"
          }
        }
      ],
      "domainParameterGenerationMode": "FB",
      "scheme": "dhStatic",
      "kasRole": "initiator",
      "l": 512,
      "iutId": "123456ABCD",
      "serverId": "434156536964",
      "kdfConfiguration": {
        "kdfType": "oneStep",
        "saltMethod": "default",
        "fixedInfoPattern": "algorithmId||l||uPartyInfo||vPartyInfo",
        "fixedInfoEncoding": "concatenation",
        "auxFunction": "KMAC-128"
      },
      "macConfiguration": {
        "macType": "KMAC-128",
        "keyLen": 128,
        "macLen": 128
      },
      "keyConfirmationDirection": "unilateral",
      "keyConfirmationRole": "provider"
    },
    {
      "tgId": 33,
      "testType": "VAL",
      "tests": [
        {
          "staticPublicServer":
"E91F8965A9B76470AF9FDB598B148F035829BF8C63E0340B7C765B2C498084511426C9D981764C9636278440
          "staticPrivateIut":
"3F4EE3CDB23C30D4497C5B35860B956734580892DC24BE9D99A843651A21532E408E05844C01FC1DE364C8F4
          "staticPublicIut":
"EC56DA2652D6401660F2E387217A6791F096701C519E4B9840E486243FC652D2B8795B366F7B4D33223EAF5E
          "tcId": 321,
```

```
          "ephemeralNonceServer":
"CB30234301F9A546FBA4B9B3170EC074F01C3783660E3B6B80906288D51CC0DF959D9CBDEEB010E8B44A1EB4
          "dkmNonceIut":
"8F40A7A0FA7C75EFC8385DFE265452900FECA59E31D18E061C8B9FD2EDE67DD9C8A3B61266C197D4E751D195
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "2F481F0007ED1C4DDA0065D354B9F4F9"
          },
          "dkm":
"83B0ACC62302F145FA2AD64D3E20DD5CCE8F872C89644598C20017BC0E4D0A3F302323BD0E162B51583B05DD
          "tag": "5FDBA0B95DC928671B70A3293535C724"
        },
        {
          "staticPublicServer":
"6EA0912ED33C93A40ECD12D99429D82CD429F3C0F078453F15096EBCF300212575B043584E8175A4271CC4B7
          "staticPrivateIut":
"240952BE39A851B771D211CD5A7B5DDA2514261EA3D5A3D3CEE9AC3CC449BCA839D4543A9ADEFD28C3892EB9
          "staticPublicIut":
"C6325324879005B1CF2D88EEBF73EDB5F24AEC4BBDEFD8A3DF1B3288557BB4B5A74A3EE63F23F7B7D4FA2BC1
          "tcId": 322,
          "ephemeralNonceServer":
"3848AB4CF013DBC117D5432514226529AE04173225FE5EF1EFC55F59707244AC939EDDFE019E4EAB6C261B7A
          "dkmNonceIut":
"82840BB766B45FABDE699E47688FE3ACEFD5B5D35130787DFC1C4FF96471C80C70B5C05E167D479D84A2DD0D
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "E7B150AC9582D5F7CA73AECC77C8732D"
          },
          "dkm":
"4D373C6C68B9A58C08DD9B4B6B3CA97FFA576FA48543AF64B1ACF65644B3C33A1D785DCE476B3B9B867738D5
          "tag": "A97C47C267A6B5AEDB06A66028E1E9B4"
        },
        {
          "staticPublicServer":
"D8923B7F6F4E53C74EC8F6E3FF98B2FC5BC602BE276006DDD91E9773D9572343A12E8074DA9B5C4BF507525B
          "staticPrivateIut":
"48513AD5D454ED2EAD15D6F28275A61E766C4A0BE725A61D78F9F9D94EE74E145B5108C3BE8474D85C18B2C1
          "staticPublicIut":
"553F8CFB7CC7176866B4A806369920D73004428AE17390E6490DD670D26E5999F0E446FE4E8341977FCF2F9C
          "tcId": 323,
          "ephemeralNonceServer":
"85F3311B9E4377F95656F286660CB10020660E3F120C6E6BEBB74456827EB9E4C2BE4A39D316B07732DD62D3
          "dkmNonceIut":
"CA6D647BE5B242505745055D8B5434797B94FCA1EC4DD8130647292F4F18EECE7FE9068D1FEDBBBC6BD9FACB
```

```
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "C0CAECADF3B8E737E923FA35689F3FF5"
          },
          "dkm":
"54AB5DACFAF26997EAACF85457FDACD24906CDF3711785F98B7ED870EB1E69D1606A11F3402A1B9E349077D1
          "tag": "D6D2E124901F3F5B6EEF90C670D85837"
        },
        {
          "staticPublicServer":
"2D795B69ECDF33DD202DA94273565A98557C76DC6B1B4C26AC2FA8C08BCCEF6DBDF271D61402E2444AB5317B
          "staticPrivateIut":
"41FC507C539F1D91A60CD5E52EA64E0F3688FCDFEEB2F6BF9EE646F52927DE36B1D2867CA4326338F92C4563
          "staticPublicIut":
"4E606053F417781025D5C30F7DB451AA6B566C29646D4A3B72C1243A181E04D9F3F0EF3C6C066A07DFFB6B0A
          "tcId": 324,
          "ephemeralNonceServer":
"AB76CD59643CDF44E4E84F8FDF75242FF6B1F20B3F9F2D5B850B6A8F4ABE43B474A7BCC86DBF44A70D8CEDF9
          "dkmNonceIut":
"61565A3B2B7F761FDC4B5DDB14B547EB6A10CF24F97BF540E0393E0596564730988E3D5AA75DEE177A5222AC
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "475558A7B67400A4B5DF86327A116FE1"
          },
          "dkm":
"3F3574B4A0103B2DD9EF38C614E08339DA3A955582CF409AE1991B62B21139EB1F79AB560E6E89F70ED4DD71
          "tag": "4A8B9E4AAAD0CC1DD7AE93C715F59AE6"
        },
        {
          "staticPublicServer":
"6D6A0CC2201B9B42AF0BF2CDFE3F912CE673CA87686995D900066F370B6431E9718996E5C52C38979C44800D
          "staticPrivateIut":
"16A8FA271986FC31DE8E1BD6EA1C40D1AE55609FE63C3531368347EF09D411CD1312EF30B73AFD12AF1ACBC0
          "staticPublicIut":
"C7EAD0576BBD6475F8E39772A92F1C69D4D24BEF3C247C5EE5E9249816A46052AC9D1373B9FA966DD3C53195
          "tcId": 325,
          "ephemeralNonceServer":
"3006316EE1E62E7437075141AB9A92A0658D2874E588EABA69C483E6CB0D3ED6FC50B6D3895DED2036E828F7
          "dkmNonceIut":
"1B3A58C69BC49434A4BA9460C2A1C4283B243913796948689C814359B99540EC1647FE6571AB193A3792AB30
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "00241841E800477D50AF88F2BFFA94F5"
```

```
        },
        "dkm":
"B635A63986441D5690DC3D4349530D3534A90D9646B50FB5BF6EE8C26A85EF9F2AA9750FC6CA26226C485BC6
        "tag": "83C7EC97C3DFCF35B1D8AD7070435591"
      }
    ],
    "domainParameterGenerationMode": "ffdhe2048",
    "scheme": "dhStatic",
    "kasRole": "initiator",
    "l": 512,
    "iutId": "123456ABCD",
    "serverId": "434156536964",
    "kdfConfiguration": {
      "kdfType": "oneStep",
      "saltMethod": "default",
      "fixedInfoPattern": "algorithmId||l||uPartyInfo||vPartyInfo",
      "fixedInfoEncoding": "concatenation",
      "auxFunction": "KMAC-128"
    },
    "macConfiguration": {
      "macType": "KMAC-128",
      "keyLen": 128,
      "macLen": 128
    },
    "keyConfirmationDirection": "unilateral",
    "keyConfirmationRole": "provider"
  },
  {
    "p":
"9D634DCB41FC0D19295FE7624F81B808198372371D9DF1A2F15417751CB0AE3A6274D1ACECCA2F24F1704101
    "q": "F785E93BE976B687A4546D433612F4E471390E5206AF348DC073B1A9",
    "g":
"474587EA5352EAF9DDA550317C80E8041EF4830DBDDC04201768DCF845AC54BCDACBF277640461909EDE343A
    "tgId": 41,
    "testType": "VAL",
    "tests": [
      {
        "staticPublicServer":
"6E4D4A1FAD84C1ED5E0BFC8DFCCBAF61EF24D4ED1F213DA66734AFF8F2ACF151ECDF80FD93649D1069C68AB0
        "staticPrivateIut":
"A850AE4A3705D2F74E70EFFA29B2D5D2B3C90AAAA2A0889520FA0057",
        "staticPublicIut":
"47BCF083BE62C7FE3961C23C8079A4AF33B7C1864A8A407B59D4BF333821E28044BA785AC9DB493EC915D0A4
        "tcId": 521,
        "ephemeralNonceServer":
"3D6B687BBC83821531FFB77A770F0FE32E4423E850C6BA606CC2F39791E8C42CE756DDFB0F8954C9C6738455
```

```
          "dkmNonceIut":
"94D4E9B53B1E6D08CFF6D98EA6763B1908FD221EE5145295E2106BB5",
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "7CF174DEFCD27FEDB7CEBA8A03B8D994"
          },
          "dkm":
"5CAD00C6D5552E40154113160CCC6AF812ACF834BD2D94900F6BE94505662A86EA8B6F3515D1ACA8779F6F7C
          "tag": "6161E58FAAE568690265B605AF9803A3"
        },
        {
          "staticPublicServer":
"33690FB54CE32790C4D72FDE93EF47E147C2BC1F177879508EC7C28611302F82E27C23C6A45DA40E77A24BBD
          "staticPrivateIut":
"D36B419AAA6F3405D923495288A8F68849A74D169C7C71F565E580F4",
          "staticPublicIut":
"27AFB6287C93CA74D51AC1A14F062CBF2CF628461920476181BE7C09D876FA0BBA52D87949F21F446A995031
          "tcId": 522,
          "ephemeralNonceServer":
"87921CEFA918B2DDEE56C490252839C7550C46591BE2BFA5FCCE6E2D11FA05EC4264BDD47C477C824FA35896
          "dkmNonceIut":
"75D7A7980D2C9ED413F2F768FD39DCBC5578807C748AFA17FC3B3080",
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "2FAB3DB49A83B8B53FBB22C01AA9BE0E"
          },
          "dkm":
"BFB09F4326253334894AAFA9A405CBC6CC6004461C2639C2B8FF8311E0956491EA5E8EF0E79EB7BE2929B9A4
          "tag": "9FA3D8D05E478FD8D090B9F1E49C6612"
        },
        {
          "staticPublicServer":
"1F4B401F25A02FC8431C01DCD556C40C2F3552A7A57E041966BBC06B9234E46390820C5658C4728C3851C550
          "staticPrivateIut":
"2CCF7352D26B2AA83872E244D6E170B3BD0DFA026AB4E33B8F72DA55",
          "staticPublicIut":
"07B03DF8FDA5C8DEF7BF71B1F801A5FE754779CAE34FCD9FEC8A9052DBEEE7FACD33762079E09D4B30B021E4
          "tcId": 523,
          "ephemeralNonceServer":
"22AF3CF6F39A38BF65A70757E81EB4D31E66B57CAA4B71D7D0E42F9FE199A61C47420BA4B4CC89E6A1D5740B
          "dkmNonceIut":
"F0B7A8233387BB319716CF645115F66A9015ADE5F16E563CCA5E81FF",
          "kdfParameter": {
            "kdfType": "oneStep",
```

```
            "salt": "00000000000000000000000000000000",
            "algorithmId": "2D3A39C8E35875E6755EA52515C18A4D"
          },
          "dkm":
"1A12C430737A078ACAC646E425C60AD4B2666B320EC8BB5A5F42368C83A817925A305E5BE3EE92B234BDDA1E
          "tag": "55F7B5A17C260CF3A157429AF4B59947"
        },
        {
          "staticPublicServer":
"65894C01939E51A35A6198A9C53112262586692C00E824E02F4D5B32D24C4479CC0A73F9E92871A07CA4EB11
          "staticPrivateIut":
"E92DD17D6D9711241F47409E7A9778A47F9234FBDFD422DE448E6906",
          "staticPublicIut":
"0AF605B073971E58B1DDFEB163D0E998F1231A01D554AED884C051511C24C8EBBE777C2A2AB0A2F033986CA7
          "tcId": 524,
          "ephemeralNonceServer":
"BC868BEDB9EB5FD130E29BB66E5E2CF5129EC2B6E93F901D44BF0897D3B30F2B81DF1387446E0A714B9FBDF2
          "dkmNonceIut":
"1928FC8E38CC9B3BE4F75273EEDFF28C0AD30064533EFC70FC400F04",
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "480BABBBAF2151F3C385735CFA77C1CF"
          },
          "dkm":
"0166D0CFF75891DB504375C0240E3D5A3642E92F88490187444F8F49E3B7403CBAA759D2054F550357A8A5B7
          "tag": "4A655FC435F02FA051E61963C3E096BA"
        },
        {
          "staticPublicServer":
"2BFFCC25FC6BA96B1A0B55C47167A30FD63381B4FEADFA5306DEE7945C6D25B35444A9E44B23CB94B5F29598
          "staticPrivateIut":
"F0D656FC4F062F37B3E33D0B87FA2C2ED673D42A43DD3DB3E13CF523",
          "staticPublicIut":
"87DAD28E0F42B3FA82187ADDE52572C48CB8444A7F407BC27FFCFECD338ABBD25725884834097A52C9BBA272
          "tcId": 525,
          "ephemeralNonceServer":
"4DC1D1E3A776BD333275A306E238FD26108C385C5B9BAAE1EAC1534CBB7BD41651D268A55AA5A6491116DC57
          "dkmNonceIut":
"75DD423E532939DBE21E199D4E8D650878F0C855C4527E2B5F73028B",
          "kdfParameter": {
            "kdfType": "oneStep",
            "salt": "00000000000000000000000000000000",
            "algorithmId": "961E859DB07CD52017857D32F3C33E3A"
          },
```

```
          "dkm":
"D42BDEE9069D4AFEE6FA9183FF8FE2398ACD4E2744BBAB5C407D4B779F630D29EE8B61C81371E4B2460B2FC7
          "tag": "CF846D97E7163F0D818744F05F6D8E58"
        }
      ],
      "domainParameterGenerationMode": "FB",
      "scheme": "dhStatic",
      "kasRole": "initiator",
      "l": 512,
      "iutId": "123456ABCD",
      "serverId": "434156536964",
      "kdfConfiguration": {
        "kdfType": "oneStep",
        "saltMethod": "default",
        "fixedInfoPattern": "algorithmId||l||uPartyInfo||vPartyInfo",
        "fixedInfoEncoding": "concatenation",
        "auxFunction": "KMAC-128"
      },
      "macConfiguration": {
        "macType": "KMAC-128",
        "keyLen": 128,
        "macLen": 128
      },
      "keyConfirmationDirection": "unilateral",
      "keyConfirmationRole": "provider"
    }
  ]
}
```


Figure 4

7. 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 23 — Vector Set Response JSON Object

JSON Value	Description	JSON type	Optional
acvVersion	Protocol version identifier	value	No
vsId	Unique numeric identifier for the vector set	value	No
testGroups	Array of JSON objects that represent each test vector group. See Table 24.	array	No

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 24 — Vector Set Group Response JSON Object

JSON Value	Description	JSON type	Optional
tgId	The test group Id	value	No
tests	Array of JSON objects that represent each test vector group. See Table 25.	array	No

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.

Table 25 — Vector Set Test Case Response JSON Object

JSON Value	Description	JSON type	Optional
tcId	The test case Id	value	No
testPassed	Used in VAL test types, should the KAS/ KTS negotiation have succeeded?	boolean	Yes
ephemeralPublicKeyIut	The IUT's ephemeral public key.	value	Yes
staticPublicKeyIut	The IUT's static public key.	value	Yes
dkmNonceIut	The IUT's nonce used in static schemes for Key Confirmation.	value	Yes

JSON Value	Description	JSON type	Optional
ephemeralNonceIut	The IUT's ephemeral nonce used in some schemes.	value	Yes
dkm	The derived keying material.	value	Yes
tag	The tag generated as a part of key confirmation (from the IUT perspective).	value	Yes

7.1. Example Test Results KAS-FFC JSON Object

The following is a example JSON object for KAS-FFC test results sent from the crypto module to the ACVP server.


```
[
  {
    "acvVersion": "version"
  },
  {
    "vsId": 0,
    "algorithm": "KAS-FFC",
    "revision": "Sp800-56Ar3",
    "testGroups": [
      {
        "tgId": 1,
        "tests": [
          {
            "staticPublicIut":
 "BD32AFAB7EBAF8DAAEAB8B057D9C31CAF177D2DBB8D2B423EA16F4FD72277728F50B1C0B2E4C4A1103502115
            "tcId": 1,
            "dkmNonceIut":
 "3ACB3A7D540AB3BEC07A36CAA6B60CD70601AE11A8BC963B5C8EF6E438AAF4A88FA043594E7E7B1BE6B09B80
            "dkm":
 "07F150EC5FDCD090512E0F2BE4B648B2D479567F661F4A2A9DC0043236DC5B56BA97D358F6EF6E83D6F69E75
            "tag": "ECD0FA1D72AD7C59949AB758325CBDCB"
          }
        ]
      },
      {
        "tgId": 9,
        "tests": [
          {
            "staticPublicIut":
 "7B2EE7E07764FE2CB330D52EC6BF133ECCC347320F351E2B4B9F01C11AF1709313128F0E6484FB980AA19AA0
```

```
            "tcId": 81,
            "dkmNonceIut":
"2C8A7E3FBF20906AB62D52E1BC1F92483AFE51EBA31B2A7708C80296",
            "dkm":
"CFF4F572A4BE825FC972047B414A90886E06CFFD22FB0FC30BC935F8B93FC67DD5BC94B0333A1E88D7A24D12
            "tag": "65A634025FD868396741743F5BC601F9"
          }
        ]
      },
      {
        "tgId": 33,
        "tests": [
          {
            "tcId": 321,
            "testPassed": false
          },
          {
            "tcId": 322,
            "testPassed": true
          },
          {
            "tcId": 323,
            "testPassed": true
          },
          {
            "tcId": 324,
            "testPassed": true
          },
          {
            "tcId": 325,
            "testPassed": false
          }
        ]
      },
      {
        "tgId": 41,
        "tests": [
          {
            "tcId": 521,
            "testPassed": true
          },
          {
            "tcId": 522,
            "testPassed": true
          },
          {
```

```
          "tcId": 523,
          "testPassed": true
        },
        {
          "tcId": 524,
          "testPassed": true
        },
        {
          "tcId": 525,
          "testPassed": false
        }
      ]
    }
  ]
 }
]
```


Figure 5

8. 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	2020-01-31	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.

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, 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.

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.