ACVP KAS ECC JSON Specification

Barry Fussell
Cisco Systems, Inc.
170 West Tasman Drive, San Jose, California

Russell Hammett
G2, Inc.
302 Sentinel Drive, Suite #300, Annapolis Junction, MD 20701

November 01, 2018

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

Abstract

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

2. Supported KAS-ECCs

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

- KAS-ECC / null / 1.0
- KAS-ECC / Component / 1.0
- KAS-ECC / CDH-Component / 1.0

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. 2].

3.2.1. KAS-ECC Requirements Covered

- SP 800-56Ar2—4.1 Key Establishment Preparations. The ACVP server is responsible for generating domain parameters as per the IUT's capability registration.
- SP 800-56Ar2—4.2 Key-Agreement Process. Both the ACVP server and IUT participate in the Key Agreement process. 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 800-56Ar2—5.1 Cryptographic Hash Functions. All modes of performing KAS SHALL make use of a hash function. The hash function **MAY** be used for validation of a successfully generated shared secret Z (noKdfNoKc), or as a primitive within the KDF being tested (kdfNoKc and kdfKc).
- SP 800-56Ar2—5.2 Message Authentication Code (MAC) Algorithm. A MAC is utilized for confirmation of success for kdfNoKc and kdfKc modes of KAS. Note—a MAC prerequisite is **REQUIRED** only for kdfKc, though is utilized for both kdfNoKc and kdfKc.
- SP 800-56Ar2—5.4 Nonce. Nonces are made use of in various KAS schemes—both the ACVP server and IUT SHALL be expected to generate nonces.
- SP 800-56Ar2—5.5 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 800-56Ar2 — 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. Random tests from the VAL groups, **MAY** inject bad keypairs that the IUT **MUST** be able detect. These random tests are only present in groups given appropriate assurance functions see: Section 4.4
- SP 800-56Ar2 — 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 800-56Ar2 — 5.8 Key-Derivation Methods for Key-Agreement Schemes. All schemes/modes save noKdfNoKc (component) **MUST** make use of a KDF. KDF construction **SHALL** utilize Section 4.11.1 for its pattern.
- SP 800-56Ar2 — 5.9 Key Confirmation. Most KAS schemes **MAY** allow for a Key Confirmation process, the ACVP server and IUT **MAY** be Providers or Recipients of said confirmation. Additionally, key confirmation **MAY** be performed on one or both parties (depending on scheme).
- SP 800-56Ar2 — 6 Key Agreement Schemes. All schemes specified in referenced document are supported for validation with the ACVP server.

3.2.2. KAS-ECC Requirements Not Covered

- SP 800-56Ar2 — 4.1 Key Establishment Preparations. The ACVP server **SHALL NOT** make a distinction between IUT generated keys via a trusted third party and the IUT itself.
- SP 800-56Ar2 — 5.3 Random Number Generation. The IUT **MUST** perform all random number generation with a validated random number generator. A DRBG is **REQUIRED** as a prerequisite to KAS, but **SHALL NOT** be in the scope testing assurances.
- SP 800-56Ar2 — 5.4 Nonce. Nonce generation is utilized for several schemes. The various methods of generating a nonce described in section 5.4 **MUST** be used, however their generation **SHALL NOT** be in scope of KAS testing assurances.
- SP 800-56Ar2 — 5.5.2 Assurances of Domain-Parameter Validity. The ACVP server **SHALL** generate all domain parameters, IUT validation of such parameters is **SHALL NOT** be in scope for KAS testing.
- SP 800-56Ar2 — 5.5.3 Domain Parameter Management. Domain Parameter Management **SHALL NOT** be in scope for KAS testing.
- SP 800-56Ar2 — 5.6 Key-Pair Generation. While Key-Pairs **MUST** be used in each KAS scheme, the generation, assurances, and management of said key-pairs **SHALL NOT** be in scope of KAS testing.
- SP 800-56Ar2 — 5.8 Key-Derivation Methods for Key-Agreement Schemes. Two-step Key-Derivation (Extraction-then-Expansion) **SHALL NOT** be utilized in KAS testing.

- SP 800-56Ar2 — 5.7 Rationale for Selecting a Specific Scheme. It is expected that the IUT registers all schemes it supports in its capabilities registration. Selecting specific schemes from a KAS testing perspective **SHALL NOT** be in scope.
- SP 800-56Ar2 — 8 Key Recovery. Key Recovery **SHALL NOT** be in scope of KAS testing.

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 ECC algorithms to the ACVP server.

The algorithm capabilities **MUST** be advertised as JSON objects within the 'algorithms' value of the ACVP registration message. The 'algorithms' value is an array, where each array element is an individual JSON object defined in this section. The 'algorithms' value is part of the 'capability_exchange' element of the ACVP JSON registration message. See the ACVP specification [ACVP] for more details on the registration message.

4.1. Prerequisites

Each algorithm implementation **MAY** rely on other cryptographic primitives. For example, RSA Signature algorithms depend on an underlying hash function. Each of these underlying algorithm primitives must be validated, either separately or as part of the same submission. ACVP provides a mechanism for specifying the required prerequisites:

Prerequisites, if applicable, **MUST** be submitted in the registration as the `prereqVals` JSON property array inside each element of the `algorithms` array. Each element in the `prereqVals` array **MUST** contain the following properties

Table 1 — Prerequisite Properties

JSON Property	Description	JSON Type
algorithm	a prerequisite algorithm	string
valValue	algorithm validation number	string

A "valValue" of "same" **SHALL** be used to indicate that the prerequisite is being met by a different algorithm in the capability exchange in the same registration.

An example description of prerequisites within a single algorithm capability exchange looks like this

```
"prereqVals":
[
  {
    "algorithm": "Alg1",
    "valValue": "Val-1234"
  },
  {
    "algorithm": "Alg2",
    "valValue": "same"
  }
```

]

Figure 1

4.2. Required Prerequisite Algorithms

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 — Required Prerequisite Algorithms JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
algorithm	a prerequisite algorithm	value	CCM, CMAC, DRBG, ECDSA, HMAC, SHA	valValue
algorithm validation number	value	actual number or "same"	prereqAlgVal	prerequistie algorithm validation

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

Table 3 — Prerequisite requirement conditions

Prerequisite Algorithm	Condition
DRBG	Always **REQUIRED**
SHA	Always **REQUIRED**
ECDSA	ECDSA.PKV validation **REQUIRED** when IUT using assurance functions of "fullVal", "keyPairGen", or "keyRegen". ECDSA.KeyPair validation **REQUIRED** when IUT using assurances functions of "keyPairGen", or "keyRegen".
AES-CCM	AES-CCM validation **REQUIRED** when IUT is performing KeyConfirmation (KC) and utilizing AES-CCM.
CMAC	CMAC validation **REQUIRED** when IUT is performing KeyConfirmation (KC) and utilizing CMAC.
HMAC	HMAC validation **REQUIRED** when IUT is performing KeyConfirmation (KC) and utilizing HMAC.

4.3. KAS ECC Algorithm Capabilities JSON Values

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

Table 4 — KAS ECC Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
algorithm	The algorithm under test	string	"KAS-ECC"	No
mode	The algorithm mode.	string	null, "Component", or "CDH-Component"	Yes
revision	The algorithm testing revision to use.	string	"1.0"	No
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See Section 4.2	No
function	Type of function supported	array	See Section 4.4	No
scheme	Array of supported key agreement schemes each having their own capabilities	object	See Section 4.5.1	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 ECC Functions

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

- dpGen — IUT can perform domain parameter generation (FFC only)
- dpVal — IUT can perform domain parameter validation (FFC only)
- keyPairGen — IUT can perform keypair generation.
- fullVal — IUT can perform full public key validation ([SP 800-56A Rev. 2] section 5.6.2.3.1 / 5.6.2.3.3)
- ACVP server **MAY** inject keys into "VAL" type tests that will fail full public key validation.
- partialVal — IUT can perform partial public key validation ([SP 800-56A Rev. 2] section 5.6.2.3.2 / 5.6.2.3.4)
- ACVP server **MAY** inject keys into "VAL" type tests that will fail partial public key validation.
- keyRegen — IUT can regenerate keys given a specific seed and domain parameter (pqg for FFC, curve for ECC)

4.5. KAS ECC Schemes

4.5.1. KAS ECC Scheme Capabilities JSON Values

All other scheme capabilities are advertised as a self-contained JSON object using the following values. Note that at least one of “noKdfNoKc”, “kdfNoKc”, or “kdfKc” **MUST** be supplied with the registration. See Section 4.5.2 for allowed ECC scheme types.

Table 5 — KAS ECC Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
kasRole	Roles supported for key agreement	array	initiator and/or responder	No
noKdfNoKc	Indicates no KDF, no KC tests are to be generated. Note this is a COMPONENT mode only test. This property **MUST** only be used with “KAS-ECC” / “Component”	object	Section 4.6.1	Yes
kdfNoKc	Indicates KDF, no KC tests are to be generated. Note this is a KAS-ECC only test. This mode **MAY** only be used for registrations with “KAS-ECC” (no mode)	object	Section 4.6.2	Yes
kdfKc	Indicates KDF, KC tests are to be generated. Note this is a KAS-ECC only test. This mode **MAY** only be used for registrations with “KAS-ECC” (no mode)	object	Section 4.6.3	Yes

4.5.2. Supported KAS ECC Schemes

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

- ephemeralUnified—keyConfirmation not supported
- fullMqv
- fullUnified
- onePassDh—Can only provide unilateral key confirmation party V to party U.
- onePassMqv
- onePassUnified

- staticUnified

4.6. KAS ECC Modes

4.6.1. KAS ECC noKdfNoKc

Contains properties **REQUIRED** for "noKdfNoKc" registration.

Table 6 — NoKdfNoKc Capabilities

JSON Value	Description	JSON type	Valid Values	Optional
parameterSet	The parameter sets supported	object	Section 4.7.1	No

4.6.2. KAS ECC kdfNoKc

Contains properties **REQUIRED** for "kdfNoKc" registration.

Table 7 — kdfNoKc Capabilities

JSON Value	Description	JSON type	Valid Values	Optional
kdfOption	The kdf options supported	object	Section 4.11	No
dkmNonceTypes	The dkmNonceTypes supported	array of string	randomNonce, timestamp, sequence, timestampSequence	Required for staticUnified scheme
parameterSet	The parameter sets supported	object	Section 4.7.1	No

4.6.3. KAS ECC kdfKc

Contains properties **REQUIRED** for "kdfKc" registration.

Table 8 — kdfKc Capabilities

JSON Value	Description	JSON type	Valid Values	Optional
kdfOption	The kdf options supported	object	Section 4.11	No
dkmNonceTypes	The dkmNonceTypes supported	array of string	randomNonce, timestamp, sequence, timestampSequence	Required for staticUnified scheme
kcOption	The kc options supported	object	Section 4.12	No
parameterSet	The parameter sets supported	object	Section 4.7.1	No

4.7. Parameter Sets

4.7.1. KAS ECC Parameter Set

Each parameter set advertised is a self-contained JSON object using the following values. Note that at least one parameter set ("eb", "ec", "ed", "ee") is **REQUIRED**.

Table 9 — KAS ECC Parameter Set Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
eb	The eb parameter set	object	See Section 4.7.2	Yes
ec	The ec parameter set	object	See Section 4.7.2	Yes
ed	The ed parameter set	object	See Section 4.7.2	Yes
ee	The ee parameter set	object	See Section 4.7.2	Yes

4.7.2. KAS ECC Parameter Set Details

- eb: Len n—224-255, min Len h—112, min hash len—112, min keySize—112, min macSize—64
- ec: Len n—256-283, min Len h—128, min hash len—128, min keySize—128, min macSize—64
- ed: Len n—384-511, min Len h—192, min hash len—192, min keySize—192, min macSize—64
- ee: Len n—512+, min Len h—256, min hash len—256, min keySize—256, min macSize—64

"noKdfNoKc" **REQUIRES** "hashAlg"

"kdfNoKc" **REQUIRES** "hashAlg" and at least one valid MAC registration

"kdfKc" **REQUIRES** "hashAlg" and at least one valid MAC registration

Table 10 — KAS ECC Parameter Set Details Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
curve	The elliptic curve to use for key generation.	value	See Section 4.8	No
hashAlg	The hash algorithms to use for KDF (and noKdfNoKc)	array	See Section 4.9	No
macOption	The macOption(s) to use with "kdfNoKc" and/or "kdfKc"	object	See Section 4.10	Yes

4.8. Supported ECC Curves

The following ECC Curves **MAY** be advertised by the ACVP compliant crypto module:

Table 11 — Supported Curves per parameter set.

Parameter Set	Prime	Koblitz	Binary
eb	P-224	K-233	B-233
ec	P-256	K-283	B-283
ed	P-384	K-409	B-409
ee	P-521	K-571	B-571

4.9. Supported Hash Algorithm Methods

The following SHA methods **MAY** be advertised by the ACVP compliant crypto module:

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

4.10. Supported KAS ECC MAC Options

The following MAC options **MAY** be advertised for registration under a “kdfNoKc” and “kdfKc” kasMode:

- AES-CCM
- CMAC
- HMAC-SHA-1
- HMAC-SHA2-224
- HMAC-SHA2-256
- HMAC-SHA2-384
- HMAC-SHA2-512

Table 12 — KAS ECC Mac Option Details

JSON Value	Description	JSON type	Valid Values	Optional
keyLen	The supported keyLens for the selected MAC.	Domain	AES based MACs limited to 128, 192, 256. HashAlg based MACs mod 8. All keySizes minimum **MUST** conform to parameter set requirements See Section 4.7.2 .	No

JSON Value	Description	JSON type	Valid Values	Optional
nonceLen	The nonce len for use with AES-CCM mac	value	Input as bits, 56-104, odd byte values only (7-13). Additionally minimum **MUST** conform to parameter set requirements See Section 4.7.2 .	Yes (required for AES-CCM)
macLen	The mac len for use with mac	value	Input as bits, mod 8, minimum **MUST** conform to parameter set requirements See Section 4.7.2 , maximum **SHALL NOT** exceed block size. .	No

4.11. Supported KAS ECC KDF Options

The following MAC options are available for registration under a "kdfNoKc" and "kdfKc" kasMode:

- concatenation

Table 13 — KAS ECC KDF Option Details

JSON Value	Description	JSON type	Valid Values	Optional
oiPattern	The OI pattern to use for constructing OtherInformation.	value	See Section 4.11.1 .	No

4.11.1. Other Information Construction

Some IUTs **MAY** require a specific pattern for the OtherInfo portion of the KDFs for KAS. An "oiPattern" is specified in the KDF registration to accommodate such requirements. Regardless of the oiPattern specified, the OI bitlength **MUST** be 240 for FFC, and 376 for ECC. The OI **SHALL** be padded with random bits (or the most significant bits utilized) when the specified OI pattern does not meet the bitlength requirement

Pattern candidates:

- literal[123456789ABCDEF]
    - uses the specified hex within "[]". literal[123456789ABCDEF] substitutes "123456789ABCDEF" in place of the field

- uPartyInfo
  - uPartyId { || ephemeralKey } { || ephemeralNonce } { || dkmNonce }
    - dkmNonce is provided by party u for static schemes
    - “optional” items such as ephemeralKey **MUST** be included when available for ACVP testing.
- vPartyInfo { || ephemeralKey } { || ephemeralNonce }
  - vPartyId
    - “optional” items such as ephemeralKey **MUST** be included when available for ACVP testing.
- counter
  - 32bit counter starting at “1” (0×00000001)

Example (Note that party U is the server in this case “434156536964”, party V is the IUT “a1b2c3d4e5”, using an ECC non-static scheme):

- “concatenation” : “literal[123456789CAFECAFE]||uPartyInfo||vPartyInfo”

Evaluated as:

- “123456789CAFECAFE434156536964a1b2c3d4e5b16c5f78ef56e8c14a561”
  - “b16c5f78ef56e8c14a561” are random bits applied to meet length requirements

4.12. Supported KAS ECC KC Options

The following KC options are available for registration under a “kdfKc” kasMode:

Table 14 — KAS ECC KC Option Details Capabilities

JSON Value	Description	JSON type	Valid Values	Optional
kcRole	The role(s) the IUT is to act as for KeyConfirmation.	array	provider/recipient	No
kcType	The type(s) the IUT is to act as for KeyConfirmation.	array	unilateral/bilateral	No
nonceType	The nonce type(s) the IUT is to use for KeyConfirmation.	array	randomNonce, timestamp, sequence, timestampSequence	No

4.13. Example KAS ECC Capabilities JSON Object

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

```
{
```

```
"algorithm": "KAS-ECC",
"revision": "1.0",
"prereqVals": [{
  "algorithm": "ECDSA",
  "valValue": "123456"
 },
 {
  "algorithm": "DRBG",
  "valValue": "123456"
 },
 {
  "algorithm": "SHA",
  "valValue": "123456"
 },
 {
  "algorithm": "CCM",
  "valValue": "123456"
 },
 {
  "algorithm": "CMAC",
  "valValue": "123456"
 },
 {
  "algorithm": "HMAC",
  "valValue": "123456"
 }
],
"function": ["keyPairGen", "dpGen"],
"scheme": {
 "ephemeralUnified": {
  "kasRole": ["initiator", "responder"],
  "kdfNoKc": {
   "kdfOption": {
    "concatenation": "uPartyInfo||vPartyInfo",
    "ASN1": "uPartyInfo||vPartyInfo"
   },
   "parameterSet": {
    "ec": {
     "curve": "K-283",
     "hashAlg": ["SHA2-224", "SHA2-256"],
     "macOption": {
      "AES-CCM": {
       "keyLen": [128],
       "nonceLen": 56,
       "macLen": 64
      }
```

```
          }
        }
      }
    }
  }
 }
}
```

Figure 2

4.14. Example KAS ECC Component Capabilities JSON Object

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

```
{
 "algorithm": "KAS-ECC",
 "mode": "Component",
 "revision": "1.0",
 "prereqVals": [{
   "algorithm": "ECDSA",
   "valValue": "123456"
  },
  {
   "algorithm": "DRBG",
   "valValue": "123456"
  },
  {
   "algorithm": "SHA",
   "valValue": "123456"
  },
  {
   "algorithm": "CCM",
   "valValue": "123456"
  },
  {
   "algorithm": "CMAC",
   "valValue": "123456"
  },
  {
   "algorithm": "HMAC",
   "valValue": "123456"
  }
 ],
 "function": ["keyPairGen", "dpGen"],
 "scheme": {
  "ephemeralUnified": {
   "kasRole": ["initiator", "responder"],
```

```
   "noKdfNoKc": {
    "parameterSet": {
     "eb": {
      "curve": "P-224",
      "hashAlg": ["SHA2-224", "SHA2-256"]
     }
    }
   }
  }
 }
}
```


Figure 3

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 15 — Required Party Generation Obligations

Scheme	KasMod	KasRole	KeyConfirmation	KeyConfirmationDirect	StaticKeyP	EphemeralKey	EphemeralNo	DkmNon
fullUnified	NoKdfNoKc	InitiatorPartyU	None	None	True	True	False	False
fullUnified	NoKdfNoKc	ResponderPartyV	None	None	True	True	False	False
fullUnified	KdfNoKc	InitiatorPartyU	None	None	True	True	False	False
fullUnified	KdfNoKc	ResponderPartyV	None	None	True	True	False	False
fullUnified	KdfKc	InitiatorPartyU	Provider	Unilateral	True	True	False	False
fullUnified	KdfKc	InitiatorPartyU	Provider	Bilateral	True	True	False	False
fullUnified	KdfKc	InitiatorPartyU	Recipient	Unilateral	True	True	False	False
fullUnified	KdfKc	InitiatorPartyU	Recipient	Bilateral	True	True	False	False
fullUnified	KdfKc	ResponderPartyV	Provider	Unilateral	True	True	False	False
fullUnified	KdfKc	ResponderPartyV	Provider	Bilateral	True	True	False	False
fullUnified	KdfKc	ResponderPartyV	Recipient	Unilateral	True	True	False	False
fullUnified	KdfKc	ResponderPartyV	Recipient	Bilateral	True	True	False	False
fullMqv	NoKdfNoKc	InitiatorPartyU	None	None	True	True	False	False
fullMqv	NoKdfNoKc	ResponderPartyV	None	None	True	True	False	False
fullMqv	KdfNoKc	InitiatorPartyU	None	None	True	True	False	False
fullMqv	KdfNoKc	ResponderPartyV	None	None	True	True	False	False
fullMqv	KdfKc	InitiatorPartyU	Provider	Unilateral	True	True	False	False
fullMqv	KdfKc	InitiatorPartyU	Provider	Bilateral	True	True	False	False
fullMqv	KdfKc	InitiatorPartyU	Recipient	Unilateral	True	True	False	False
fullMqv	KdfKc	InitiatorPartyU	Recipient	Bilateral	True	True	False	False
fullMqv	KdfKc	ResponderPartyV	Provider	Unilateral	True	True	False	False
fullMqv	KdfKc	ResponderPartyV	Provider	Bilateral	True	True	False	False
fullMqv	KdfKc	ResponderPartyV	Recipient	Unilateral	True	True	False	False
fullMqv	KdfKc	ResponderPartyV	Recipient	Bilateral	True	True	False	False
ephemeralUnified	NoKdfNoKc	InitiatorPartyU	None	None	False	True	False	False
ephemeralUnified	NoKdfNoKc	ResponderPartyV	None	None	False	True	False	False
ephemeralUnified	KdfNoKc	InitiatorPartyU	None	None	False	True	False	False
ephemeralUnified	KdfNoKc	ResponderPartyV	None	None	False	True	False	False
onePassUnified	NoKdfNoKc	InitiatorPartyU	None	None	True	True	False	False
onePassUnified	NoKdfNoKc	ResponderPartyV	None	None	True	False	False	False
onePassUnified	KdfNoKc	InitiatorPartyU	None	None	True	True	False	False
onePassUnified	KdfNoKc	ResponderPartyV	None	None	True	False	False	False
onePassUnified	KdfKc	InitiatorPartyU	Provider	Unilateral	True	True	False	False
onePassUnified	KdfKc	InitiatorPartyU	Provider	Bilateral	True	True	False	False
onePassUnified	KdfKc	InitiatorPartyU	Recipient	Unilateral	True	True	False	False
onePassUnified	KdfKc	InitiatorPartyU	Recipient	Bilateral	True	True	False	False

Scheme	KasMode	KasRole	KeyConfirmationRole	KeyConfirmationDirection	StaticKeyPair	EphemeralKeyPair	EphemeralNonce	DkmNonce
onePassUnified	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
onePassUnified	KdfKc	ResponderPartyV	Provider	Bilateral	True	False	True	False
onePassUnified	KdfKc	ResponderPartyV	Recipient	Unilateral	True	False	True	False
onePassUnified	KdfKc	ResponderPartyV	Recipient	Bilateral	True	False	True	False
onePassMqv	NoKdfNoKc	InitiatorPartyU	None	None	True	True	False	False
onePassMqv	NoKdfNoKc	ResponderPartyV	None	None	True	False	False	False
onePassMqv	KdfNoKc	InitiatorPartyU	None	None	True	True	False	False
onePassMqv	KdfNoKc	ResponderPartyV	None	None	True	False	False	False
onePassMqv	KdfKc	InitiatorPartyU	Provider	Unilateral	True	True	False	False
onePassMqv	KdfKc	InitiatorPartyU	Provider	Bilateral	True	True	False	False
onePassMqv	KdfKc	InitiatorPartyU	Recipient	Unilateral	True	True	False	False
onePassMqv	KdfKc	InitiatorPartyU	Recipient	Bilateral	True	True	False	False
onePassMqv	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
onePassMqv	KdfKc	ResponderPartyV	Provider	Bilateral	True	False	True	False
onePassMqv	KdfKc	ResponderPartyV	Recipient	Unilateral	True	False	True	False
onePassMqv	KdfKc	ResponderPartyV	Recipient	Bilateral	True	False	True	False
onePassDh	NoKdfNoKc	InitiatorPartyU	None	None	False	True	False	False
onePassDh	NoKdfNoKc	ResponderPartyV	None	None	True	False	False	False
onePassDh	KdfNoKc	InitiatorPartyU	None	None	False	True	False	False
onePassDh	KdfNoKc	ResponderPartyV	None	None	True	False	False	False
onePassDh	KdfKc	InitiatorPartyU	Recipient	Unilateral	False	True	False	False
onePassDh	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
staticUnified	NoKdfNoKc	InitiatorPartyU	None	None	True	False	False	False
staticUnified	NoKdfNoKc	ResponderPartyV	None	None	True	False	False	False
staticUnified	KdfNoKc	InitiatorPartyU	None	None	True	False	False	True
staticUnified	KdfNoKc	ResponderPartyV	None	None	True	False	False	False
staticUnified	KdfKc	InitiatorPartyU	Provider	Unilateral	True	False	False	True
staticUnified	KdfKc	InitiatorPartyU	Provider	Bilateral	True	False	False	True
staticUnified	KdfKc	InitiatorPartyU	Recipient	Unilateral	True	False	False	True
staticUnified	KdfKc	InitiatorPartyU	Recipient	Bilateral	True	False	False	True
staticUnified	KdfKc	ResponderPartyV	Provider	Unilateral	True	False	False	False
staticUnified	KdfKc	ResponderPartyV	Provider	Bilateral	True	False	True	False
staticUnified	KdfKc	ResponderPartyV	Recipient	Unilateral	True	False	True	False
staticUnified	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 SP800-56a KAS ECC 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 16 — 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 4

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 ECC is as follows:

Table 17 — 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
scheme	The scheme for the test vectors. See Section 4.5.1 for possible values	value	No
testType	The type of testCases expected within the group. AFT (Functional) tests produce test cases where the prompt file delivers only the needed public server information in which the IUT is expected to perform KAS. VAL (Validity) tests produce inputs/outputs from both server and IUT perspectives of a KAS negotiation. The expectation of the IUT on such tests is to determine if the KAS negotiation was successful or not.	AFT, VAL	No
kasRole	The KAS role	initiator, responder	No
kasMode	The KAS mode	noKdfNoKc, kdfNoKc, kdfKc	No
parmSet	Parameter set value to use	eb, ec, ed, ee	No
hashAlg	hashAlg values being used	See Section 4.9	No
macType	The MAC being used. **REQUIRED** for "kdfNoKc" and "kdfKc" modes.	See Section 4.10	Yes
keyLen	The key length of the MAC. **REQUIRED** for "kdfNoKc" and "kdfKc" modes.	See Section 4.10	Yes
nonceAesCcmLen	The nonce length of the MAC (applies only to AES-CCM). **REQUIRED** for "kdfNoKc" and "kdfKc"	See Section 4.10	Yes

	modes using a AES-CCM MAC.		
macLen	The mac length. **REQUIRED** for “kdfNoKc” and “kdfKc” modes.	See Section 4.10	Yes
kdfType	The KDF being used. **REQUIRED** for “kdfNoKc” and “kdfKc” modes.	concatenation, asn1	Yes
idServerLen	The length of the server ID. **REQUIRED** for “kdfNoKc” and “kdfKc” modes.	value	Yes
idServer	The server ID. **REQUIRED** for “kdfNoKc” and “kdfKc” modes.	value	Yes
idIutLen	The length of the server ID. **REQUIRED** for “kdfNoKc” and “kdfKc” modes. Provided in response by IUT for AFT tests.	value	Yes
idIut	The server ID. **REQUIRED** for “kdfNoKc” and “kdfKc” modes. Provided in response by IUT for AFT tests.	value	Yes
oiPattern	The oiPattern used in the KDF. **REQUIRED** for “kdfNoKc” and “kdfKc” modes.	See Section 4.11.1	Yes
kcRole	Key confirmation roles supported. **REQUIRED** for “kdfKc” modes.	provider, recipient	Yes
kcType	Key confirmation types supported. **REQUIRED** for “kdfKc” modes.	unilateral and/or bilateral	Yes
curve	The curve useds for keypair generation	value	No
tests	Array of individual test vector JSON objects,	array	No

	which are defined in Section 6.2		

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 test vector.

Table 18 — 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
staticPublicServerX	The ECDSA static public key X coordinate	value	Yes
staticPublicServerY	The ECDSA static public key Y coordinate	value	Yes
ephemeralPublicServerX	The ECDSA ephemeral public key X coordinate	value	Yes
ephemeralPublicServerY	The ECDSA ephemeral public key Y coordinate	value	Yes
nonceEphemeralServer	nonceEphemeralServer ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field	value	Yes
nonceNoKc	The 16 byte nonce concatenated to the “Standard Test Message”. Used for No Key Confirmation tests only.	value	Yes
nonceDkm	The nonce supplied by the initiator to be used in the OI field in the PartyUInfo field.	value	Yes
staticPrivateIut	The IUT ECDSA static private key	value	Yes
staticPublicIutX	The IUT ECDSA static public key X coordinate	value	Yes
staticPublicIutY	The IUT ECDSA static public key Y coordinate	value	Yes
ephemeralPrivateIut	The IUT ECDSA ephemeral private key	value	Yes
ephemeralPublicIutX	The IUT ECDSA ephemeral public key X coordinate	value	Yes
ephemeralPublicIutY	The IUT ECDSA ephemeral public key Y coordinate	value	Yes

JSON Value	Description	JSON type	Optional
oiLen	Length of the OtherInfo field	value	Yes
oi	OtherInfo field	value	Yes
dkm	Derived Keying Material.	value	Yes
tagIut	The tag (or MAC) GENERATED BY THE SERVER/IUT by using the DKM to MAC the Message with the specified method	value	Yes
nonceEphemeralIut	nonceEphemeralIut ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field	value	Yes
nonceDkmIut	ONLY USED BY STATIC SCHEME. The nonce supplied by the initiator to be used in the OI field in the PartyUInfo field	value	Yes
nonceLenDkm	ONLY USED BY STATIC SCHEME. The length of the nonce supplied by the initiator to be used in the OI field in the PartyUInfo field.	value	Yes
nonceEphemeralDkm	ONLY USED BY C(1,2) and C(0,2) schemes with KC. nonce to be used in the MacData field	value	Yes
nonceEphemralDkmLen	length of nonceEphemeralIut value.	value	Yes
nonceAesCcm	Nonce used by the CCM function, if CCM is used to generate the Tag.	value	Yes
macData	The message to be MAced.	value	Yes
	A shared secret that is used to derive secret keying material using a key derivation function.	value	Yes
hashZServer	The hashed shared secret, only provided in noKdfNoKc modes of operation.	value	Yes
hashZIut	The hashed shared secret, only provided in noKdfNoKc modes of operation.	value	Yes
testPassed	Pass Fail indicating if the IUT agrees with the Tag generated by the server.	boolean	Yes

6.3. Example Test Vectors JSON Object

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


```
[{
  "acvVersion": "1.0"
 },
 {
  "vsId": 1564,
  "algorithm": "KAS-ECC",
  "revision": "1.0",
  "testGroups": [
   {
                "tgId": 1,
    "scheme": "ephemeralUnified",
    "testType": "AFT",
    "kasRole": "initiator",
    "kasMode": "kdfNoKc",
    "parmSet": "ec",
    "hashAlg": "SHA2-256",
    "macType": "AES-CCM",
    "keyLen": 128,
    "aesCcmNonceLen": 64,
    "macLen": 128,
    "kdfType": "asn1",
    "idServerLen": 48,
    "idServer": "434156536964",
    "curve": "P-256",
    "tests": [{
     "tcId": 151,
     "ephemeralPublicServerX":
 "CBC9AF2F0FCE0F06643D7524DCCA96C78564BA77196C5F5F65DC0A119409A1F3",
     "ephemeralPublicServerY":
 "B619EBE85F2EC5E0A9B542CC77539D698C96CA5D0BDFCA224787C30CF971E3F4",
     "nonceNoKc": "BBDF1A42C9405B58B8329D583C437331",
     "nonceAesCcm": "FF5B0FD5F295257B"
    }]
   },
   {
                "tgId": 2,
    "scheme": "ephemeralUnified",
    "testType": "AFT",
    "kasRole": "responder",
    "kasMode": "kdfNoKc",
    "parmSet": "eb",
```

```
    "hashAlg": "SHA2-224",
    "macType": "HMAC-SHA2-224",
    "keyLen": 128,
    "macLen": 128,
    "kdfType": "asn1",
    "idServerLen": 48,
    "idServer": "434156536964",
    "curve": "P-224",
    "tests": [{
     "tcId": 161,
     "ephemeralPublicServerX":
"FFAD4CDB4293F61C2A74566FD4323A03C6BB3F9D6526D8E0506B2186",
     "ephemeralPublicServerY":
"0D614DAA05395A5FDF51BC769AEC355C9688ECEFCF2FE10E6DC1030E",
     "nonceNoKc": "BEAB1A2CB8406A7083105EC234603A80"
    }]
   },
   {
                "tgId": 3,
    "scheme": "ephemeralUnified",
    "testType": "VAL",
    "kasRole": "initiator",
    "kasMode": "kdfNoKc",
    "parmSet": "eb",
    "hashAlg": "SHA2-224",
    "macType": "HMAC-SHA2-224",
    "keyLen": 128,
    "macLen": 128,
    "kdfType": "asn1",
    "idServerLen": 48,
    "idServer": "434156536964",
    "idIutLen": 0,
    "curve": "P-224",
    "tests": [{
     "tcId": 181,
     "ephemeralPublicServerX":
"D489605D37C4F555E50D8F010BEE3165B93F7C749263C4BF3E9A4808",
     "ephemeralPublicServerY":
"23C8167ACFB24DC62D6747960330471B28DC646E04E593DBE6F8F1A4",
     "nonceNoKc": "6BBFEECEBBD5200C5FAE050526A77342",
     "ephemeralPrivateIut":
"343936401C5F88E658E2C9C47C2EB48DDE10506684D8B55027C05A15",
     "ephemeralPublicIutX":
"14AA2C1ECDC258FE8AD035E9A2872CD14466783F82F5F3F8D757133A",
     "ephemeralPublicIutY":
"8DD3D48BF9115EA5AB7A479FB1DAB0A46BCD6B4D1A306D5CAC254EC1",
```

```
    "oiLen": 376,
    "otherInfo":
"A1B2C3D4E5434156536964CAFECAFE2D822B413172BB3012AA986AFFAE95B46360E00AAD0D0548104C1F9463
    "tagIut": "5EEE5D912191984D89DF074B9A885411"
   }]
  },
  {
              "tgId": 4,
   "scheme": "ephemeralUnified",
   "testType": "VAL",
   "kasRole": "responder",
   "kasMode": "kdfNoKc",
   "parmSet": "eb",
   "hashAlg": "SHA2-224",
   "macType": "AES-CCM",
   "keyLen": 128,
   "aesCcmNonceLen": 64,
   "macLen": 128,
   "kdfType": "asn1",
   "idServerLen": 48,
   "idServer": "434156536964",
   "idIutLen": 0,
   "curve": "P-224",
   "tests": [{
    "tcId": 231,
    "ephemeralPublicServerX":
"A0457CF2F5D38B72FF1BF3A2CF4C7CE30F215B5E52A53C39193B1639",
    "ephemeralPublicServerY":
"38CA7951888E462D6C5F4E46FA953FF231F43D5A4F3FEBAEEBF3D52B",
    "nonceNoKc": "A889762176F5F02F8C1E4BBC0C669805",
    "ephemeralPrivateIut":
"5F76009454AE9158797467C297229569C6E2027D6AFC226A63489444",
    "ephemeralPublicIutX":
"1060CEE336B183738952CF13760D542E2F3AA60124D560EFA10F392C",
    "ephemeralPublicIutY":
"216EA3B35E630A1EA4A91C430E5B63306A83624F0FFD8ADFF63A380E",
    "oiLen": 376,
    "otherInfo":
"454156536964A1B2C3D4E5CAFECAFE9EF1EA2DC20EE820E7562CDD4DBCD5FD8CD57DB1F54961D8B0C83342C0
    "nonceAesCcm": "BD79B8A8D5559128",
    "tagIut": "5CC10EF2564B0CD23D746A47DB5B98A2"
   }]
  }
 ]
}
```

]

Figure 5

6.4. Example Test Vectors Component JSON Object

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


```
[{
  "acvVersion": "1.0"
 },
 {
  "vsId": 1565,
  "algorithm": "KAS-ECC",
  "mode": "Component",
  "revision": "1.0",
  "testGroups": [{
                "tgId": 1,
    "scheme": "ephemeralUnified",
    "testType": "AFT",
    "kasRole": "initiator",
    "kasMode": "noKdfNoKc",
    "parmSet": "eb",
    "hashAlg": "SHA2-224",
    "curve": "P-224",
    "tests": [{
     "tcId": 1,
     "ephemeralPublicServerX":
 "DACE4B35FD720DDD6B307777EBAFE53859C5FC2D330755B05B061CEB",
     "ephemeralPublicServerY":
 "195344DE0C79898C5C060BFACE1D24FDE1127ECF503EA04B08FFB9F1"
    }]
   }, {
                "tgId": 2,
    "scheme": "ephemeralUnified",
    "testType": "AFT",
    "kasRole": "responder",
    "kasMode": "noKdfNoKc",
    "parmSet": "eb",
    "hashAlg": "SHA2-224",
    "curve": "P-224",
    "tests": [{
     "tcId": 21,
     "ephemeralPublicServerX":
 "747EDBB8F62E1F06BD542FC2DD93169CB24DA6EF9E2FED4FE60FCBE6",
```

```
    "ephemeralPublicServerY":
"C7FB2C3C9B95E70D908B9992C8018B785F7BCD3E5967E37EFB18A422"
   }]
  },
  {
              "tgId": 3,
   "scheme": "ephemeralUnified",
   "testType": "VAL",
   "kasRole": "initiator",
   "kasMode": "noKdfNoKc",
   "parmSet": "eb",
   "hashAlg": "SHA2-224",
   "curve": "P-224",
   "tests": [{
    "tcId": 41,
    "ephemeralPublicServerX":
"866BD81E951787AA1130CB67BA48E22F8A9E7EFF0713418B4FB8A31C",
    "ephemeralPublicServerY":
"050C9E3DB4560313979FE465AC8624E93BC0D97E7C68AC589840BCF7",
    "ephemeralPrivateIut":
"0C9AE6286544FED81921E6495B946C6AF39DF90EC68379CEF2F7C69D",
    "ephemeralPublicIutX":
"CA296A5C86EC39C4EA626A8D9AB39DE5D5092FAA3AE2F241D7791497",
    "ephemeralPublicIutY":
"F768358D14A428C61A3229FB4BB752F02ECC1F54763CA98655A8412C",
    "hashZIut": "FC6268A34B63B5A82AF03A6CABE61C69CC57317E5E8C8F508FCB82D0"
   }]
  },
  {
              "tgId": 4,
   "scheme": "ephemeralUnified",
   "testType": "VAL",
   "kasRole": "responder",
   "kasMode": "noKdfNoKc",
   "parmSet": "eb",
   "hashAlg": "SHA2-224",
   "curve": "P-224",
   "tests": [{
    "tcId": 91,
    "ephemeralPublicServerX":
"7A2EBA553C4DC0E4D7A19A3648BA9713496EB462B1B7D83D375F7FFD",
    "ephemeralPublicServerY":
"5972BF3B114612AA5BBA14D0BE956DED03359F52ADDF0B9C2D0314E1",
    "ephemeralPrivateIut":
"9AEDA69CE438C6F8592CE3B8E14E92BE9143E82B3EED42CF62E45BF7",
```

```
     "ephemeralPublicIutX":
 "941DAF3C527D2B76AA907F60C208F8987681972E466529CA8BD962FD",
     "ephemeralPublicIutY":
 "F381EC5DBEA7F6EA3A09D2D75372C014C3DE3ECABBBBC00DDFB97359",
     "hashZIut": "BB61FA1DCA5D93A6FBB43317AABCAE22A3EDF7F72216516115935D4E"
    }]
   }
  ]
 }
]
```

Figure 6

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.

7.1. Vector Set Response JSON Object

Table 19 — Vector Set Response JSON Object

JSON Value	Description	JSON type
acvVersion	Protocol version identifier	value
vsId	Unique numeric identifier for the vector set	value
testGroups	Array of JSON objects that represent each test vector group. See Section 7.2	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.

7.2. Vector Set Group Response JSON Object

Table 20 — Vector Set Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group Id	value
tests	The tests associated to the group specified in tgId	value

7.3. Example Test Results JSON Object

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


```
[{
  "acvVersion": "1.0"
 },
 {
  "vsId": 1564,
  "testGroups": [{
    "tgId": 1,
    "tests": [{
     "tcId": 151,
     "nonceNoKc": "BBDF1A42C9405B58B8329D583C437331",
     "ephemeralPublicIutX":
 "F90FE5B7D5DA0F7849B0849D780143F4CC7E9F080465AA05DBD3E610D6B24763",
     "ephemeralPublicIutY":
 "1D746A8F960AE8425C63DE17618362F7040365D9168F21A0762526ECCC556084",
```

```
    "idIutLen": 40,
    "idIut": "A1B2C3D4E5",
    "oiLen": 376,
    "oi":
"A1B2C3D4E5434156536964CAFECAFEA0988C0EB862E29CBFBD0B087D3223B9052811800B2D1ADF1D42AE73BA
    "nonceAesCcm": "FF5B0FD5F295257B",
    "tagIut": "FF1ADCA06E582AD9E4A8B7FE3D7D9C28"
   }]
  },
  {
   "tgId": 2,
   "tests": [{
    "tcId": 161,
    "nonceNoKc": "BEAB1A2CB8406A7083105EC234603A80",
    "ephemeralPublicIutX":
"C5D934686BAB0E156D4F5CF1BDA7B044128C803E4C8AA2D9B0024FC0",
    "ephemeralPublicIutY":
"E2D8973A51A9CE0FA7FAD8A444ECAB518C672C65313BEE4150CFD50E",
    "idIutLen": 40,
    "idIut": "A1B2C3D4E5",
    "oiLen": 376,
    "oi":
"434156536964A1B2C3D4E5CAFECAFE9D9E4AB0A187C117158C9A234F4AEE8328714003BFED6C08A7F191E61D
    "tagIut": "77587ED9D13B811F200214FD5E1F864A"
   }]
  },
  {
   "tgId": 3,
   "tests": [{
    "tcId": 181,
    "testPassed": false
   }]
  },
  {
   "tgId": 4,
   "tests": [{
    "tcId": 231,
    "testPassed": false
   }]
  }
 ]
 }
]
```


Figure 7

7.4. Example Test Results Component JSON Object

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


```
[{
  "acvVersion": "1.0"
 },
 {
  "vsId": 1564,
  "testGroups": [{
    "tgId": 1,
    "tests": [{
     "tcId": 1,
     "ephemeralPublicIutX":
 "50471CE7F6FE2CAD6C901F85BF258E84571D3C88F59356B91DDBF286",
     "ephemeralPublicIutY":
 "5B8A7BC07BE15F28D34AA8324DEE93C715F569D3AF4820209F6452E7",
     "hashZIut": "96DCAF87127AB615896CCD0479C8BEAFD7EE111F384C962687D28ACC"
    }]
   },
   {
    "tgId": 2,
    "tests": [{
     "tcId": 21,
     "ephemeralPublicIutX":
 "3E95CE4241A63C4ECBDC12CF2A3FB9E56222C0D395885CF0B51B04F7",
     "ephemeralPublicIutY":
 "F8865F76DE98CFCFBBAD2E99A317636F48AC874847E0A489C96631EC",
     "hashZIut": "3B7721F7514C09DD38D62E72E20D0375A7B3AC5BD837A7B860BC65FA"
    }]
   },
   {
    "tgId": 3,
    "tests": [{
     "tcId": 41,
     "testPassed": false
    }]
   },
   {
    "tgId": 4,
    "tests": [{
     "tcId": 91,
     "testPassed": true
    }]
   }
```

```
    ]
  }
]
```


Figure 8

8. ECC CDH Component Test

The ECC CDH Component Test

8.1. ECC CDH Component Capabilities JSON Values

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

Table 21 — KAS ECC Component Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
algorithm	The algorithm under test	value	KAS-ECC	No
mode	The algorithm mode	value	CDH-Component	No
revision	The algorithm testing revision to use.	value	"1.0"	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
curve	Array of supported curves	array	See Section 4.8	No

8.1.1. Example KAS ECC CDH-Component Capabilities JSON Object

The following is a example JSON object advertising support for KAS ECC CDH-Component.

```
{
 "algorithm": "KAS-ECC",
 "mode": "CDH-Component",
 "revision": "1.0",
 "prereqVals": [{
  "algorithm": "ECDSA",
  "valValue": "123456"
 }],
 "function": ["keyPairGen"],
 "curve": ["P-224", "K-233", "B-233"]
}
```

Figure 9

8.2. ECC CDH Component TestVectors JSON Values

Table 22 — KAS ECC CDH Component TestVectors JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
algorithm	The algorithm under test	value	KAS-ECC	No
mode	The algorithm mode under test	value	CDH-Component	No
revision	The algorithm testing revision to use.	value	"1.0"	No
testGroups	Array of individual test group JSON objects, which are defined in Section 8.2.1	Array	Array of test group information	No

8.2.1. ECC CDH Component TestGroup JSON Values

Table 23 — KAS ECC CDH Component TestGroup JSON Values

JSON Value	Description	JSON type	Valid Values	Optional
testType	The test type expected within the group. AFT is the only valid value for ECC Component.	value	AFT	No
curve	The curve used in the test group	value	P-224, P-256, P-384, P-521, K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	No
tests	Array of individual test vector JSON objects, which are defined in Section 8.2.2	array		No

8.2.2. ECC CDH Component TestCase JSON Values

Table 24 — KAS ECC CDH Component TestCase JSON Values

JSON Value	Description	Valid Values	Optional
tcId	Numeric identifier for the test case, unique across the entire vector set.	value	No

JSON Value	Description	Valid Values	Optional
publicServerX	The X coordinate of the server's public key	value	Yes
publicServerY	The Y coordinate of the server's public key	value	Yes
publicIutX	The X coordinate of the iut's public key	value	No
publicIutY	The Y coordinate of the iut's public key	value	No
	The shared secret Z	value	No

8.2.3. Example KAS ECC CDH-Component Test Vectors JSON Object

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

```
[{
  "acvVersion": "1.0"
 },
 {
  "vsId": 1750,
  "algorithm": "KAS-ECC",
  "mode": "CDH-Component",
  "revision": "1.0",
  "testGroups": [{
    "tgId": 1,
    "testType": "AFT",
    "curve": "P-192",
    "tests": [{
     "tcId": 1,
     "publicServerX": "CAEF2CBA796BB7FC143D3EAED698C26AAE6F6F79DF3974EE",
     "publicServerY": "03ED6D7A90637629DBCEBFF4A2D1D771D9D4CF9F0D88CE90"
    }]
   },
   {
    "tgId": 2,
    "testType": "AFT",
    "curve": "K-163",
    "tests": [{
     "tcId": 26,
     "publicServerX": "048C46D674E1218D0BD3C9FCD120ECE8B4DB7310E7",
     "publicServerY": "ED3EEDB656E035C779081090BE44B743E857E3B4"
    }]
   },
   {
    "tgId": 3,
```

```
    "testType": "AFT",
    "curve": "B-163",
    "tests": [{
     "tcId": 51,
     "publicServerX": "8EE7C8F08BF47B21CA2FE911B721651B90E52391",
     "publicServerY": "0461DF3646E95598EAE4F5C6A634E71006ABC6FE1F"
    }]
   }
  ]
 }
]
```


Figure 10

8.3. KAS CDH-Component 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.

8.3.1. CDH Component Vector Set Response JSON Object

Table 25 — CDH Component Vector Set Response JSON Object

JSON Value	Description	JSON type
acvVersion	Protocol version identifier	value
vsId	Unique numeric identifier for the vector set	value
testGroups	Array of JSON objects that represent each test vector group. See Section 8.3.2	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.

8.3.2. CDH Component Vector Set Group Response JSON Object

Table 26 — CDH Component Vector Set Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group Id	value tests

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 DRBG test vector.

8.3.3. CDH Component Test Case Results JSON Object

Table 27 — CDH Component Test Case Results JSON Object

JSON Value	Description	JSON type	Optional
tcId	Numeric identifier for the test case, unique across the entire vector set.	value	No
publicIutX	x value of the IUT public key	value	No
publicIutY	x value of the IUT public key	value	No
	Computed shared secret Z	value	No

8.4. Example KAS ECC CDH Component Test Results JSON Object

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


```
[{
  "acvVersion": "1.0"
 },
 {
  "vsId": 1750,
  "testGroups": [{
    "tgId": 1,
    "tests": [{
     "tcId": 1,
     "publicIutX": "DB9FBC84CBAD3EED42C31CDBF2882041634D040219C3E47A",
     "publicIutY": "9BD672733BCCEF2BD805E97FF9BBFE0FFC003BEEEF56868B",
     "z": "8BEAEA60DFAC075F9F25A5CFEA39818D98D3EA4B9D4C34A8"
    }]
   },
   {
    "tgId": 2,
    "tests": [{
     "tcId": 26,
     "publicIutX": "058C593D1D4E8238102BDE6B497218D92F8EDD2997",
     "publicIutY": "0437682E4608984EFC7FB619FB260EF27CAF704D7B",
     "z": "075D9A831E0665521D613AEAA59B8C8CDFBAC8C683"
    }]
   },
   {
    "tgId": 3,
    "tests": [{
     "tcId": 51,
     "publicIutX": "04128CD094F6988AA26DA2B100A71A31214CC9C50B",
     "publicIutY": "01A3A88C9F0987E488922573D0A31D300532F0B268",
     "z": "07EC896621BF1703EB7567196ED1DE5742C4695990"
```

```
        } ]
      }
    ]
  }
]
```

Figure 11

9. 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	2018-11-01	Initial Release

Appendix D — References

S. Bradner (March 1997) *Key words for use in RFCs to Indicate Requirement Levels* (Internet Engineering Task Force), BCP 14, March 1997. RFC 2119. RFC RFC2119. DOI 10.17487/RFC2119. https://www.rfc-editor.org/info/rfc2119.

P. Hoffman (December 2016) *The "xml2rfc" Version 3 Vocabulary* (Internet Engineering Task Force), RFC 7991, December 2016. RFC 7991. RFC RFC7991. DOI 10.17487/RFC7991. https://www.rfc-editor.org/info/rfc7991.

B. Leiba (May 2017) *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words* (Internet Engineering Task Force), BCP 14, May 2017. RFC 8174. RFC RFC8174. DOI 10.17487/RFC8174. https://www.rfc-editor.org/info/rfc8174.

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, Miles E. Smid (May 2013) *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography* (Gaithersburg, MD), May 2013. SP 800-56A Rev. 2. https://doi.org/10.6028/NIST.SP.800-56Ar2.

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.