Automated Cryptographic Validation Protocol (ACVP) JSON Specification

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

Apostol Vassilev
Harold Booth
Information Technology Laboratory
Computer Security Division

January 01, 2019

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

Abstract

The ACV Protocol provides a method for communication between a cryptographic module that is embedded inside of a device or otherwise running on a platform accessible via computer network, and an external testing system, using standard network communication interfaces and protocols. This communication protocol can also be used to validate the correctness of the algorithm implementations in the cryptographic module with a validation authority.

Keywords

The following are keywords to be used by search engines and document catalogues.

ACVP; cryptography

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. It introduces a method to perform cryptographic assessment and validations at a rate which meets typical industry development cycles; providing the ability to deploy validated crypto with CVE fixes much faster than previous methods.

The ACVP specification describes how the protocol is structured with respect to the client-server model, messaging, optional features, and flows. It 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.

Unresolved directive in draft-fussell-acvp-spec.adoc — include::/__w/ACVP-standalone/ACVP-standalone/src/common/common-sections/02-conventions.adoc[]

2. Overview

ACVP has the following goals:

- To work in situations where the testing system is remote from the cryptographic module, e.g. running as a process on a separate device.
- To enable automated testing that can take place with a minimum of human interaction.
- To enable the testing system to discover the capabilities of the module being tested; that is, the particular algorithms and parameters that the module supports.
- To provide extensibility that can be used to introduce tests for new algorithms, and new tests for existing algorithms.
- To be compatible with emerging automated validation systems wherever possible, especially the FIPS-140 Cryptographic Algorithm Validation Program.
- To provide a standard communication method so that vendors can utilize the same testing service for FIPS-140, 3rd party crypto verification and product FCS readiness testing.

ACVP defines how to communicate a request (to execute a cryptographic operation) to a cryptographic module, and how to communicate the corresponding response (containing the output of that operation) back to a testing system. It defines a transport (based on HTTP or HTTPS [RFC 7230]), an JSON message structure(which is negotiated), and a set of message exchanges. Each vector test set corresponds to a single message exchange driven from the client associated with the module under test. ACVP does not define the cryptographic algorithms, nor does it detail the precise conditions for a response to be acceptable. Instead, it references existing specifications for those algorithms, and defines a mapping between the data on the wire and the algorithm testing specification. ACVP does not define detailed conformance criteria, such as those in FIPS-140. Instead, it aims to be independent of particular conformance criteria, so that it can be used in multiple domains with different (even potentially conflicting) conformance criteria. ACVP does not define an interface that can be used to manage or control a cryptographic module.

2.1. Audience

This document is written to address multiple audiences:

- Crypto module developers who require validation testing
- Crypto module developers who require runtime crypto assessment testing
- Crypto validation organizations who will perform validation testing
- Crypto module customers that desire validation testing results or verifiable artifacts of testing

2.2. Goals

The goals for this document are to provide a messaging protocol that can be used with existing authentication and communication protocols to provide a way to test crypto modules. The following functions are outside the scope of this document:

- The API to the cryptographic module
- How the tests are generated
- How the results/artifacts are stored or managed
- Authentication used
- Scalability
- Management interface

With that in mind there are several expectations when building a server used as a validation authority. A validation authority **SHALL** use a combination of HTTPS [RFC 7230], TLS 1.2 [RFC 5246] or greater and mutual authentication. Therefore a client that expects to be used with a validation authority **SHALL** have the same requirements. A server, proxy or client developed for the purposes of internal organizational testing only **MAY** choose not to include some of those features.

3. Architecture

A server/client/proxy model is used where the roles are defined as:

- ACV Client — Communicates with the ACV server using Java Script Object Notation (JSON [RFC 7159]) and collects the test vectors and returns the test results using product specific methods.
- ACV Server — Sends JSON formatted messaging and test data to the ACV client and processes test responses.
- ACV Proxy — Resides between the ACV server and ACV client to proxy the connection for the client. This is particularly useful when the client does not support TLS, key management or have signature capabilities and they are required by the server. An example architecture is provided in Figure 3.
- Device Under Test — Contains the crypto module under test which can include various algorithms and functions that encrypt/decrypt, generate keys, signatures, perform verifications and DRBG functions. May also contain the ACVP client.
- Cryptographic Module API — This is the interface, manual or otherwise, to the crypto module. This interface is environment specific and will vary depending on the crypto module and may not be limited to real-time operation.

3.1. Server/Client Architecture(realtime)

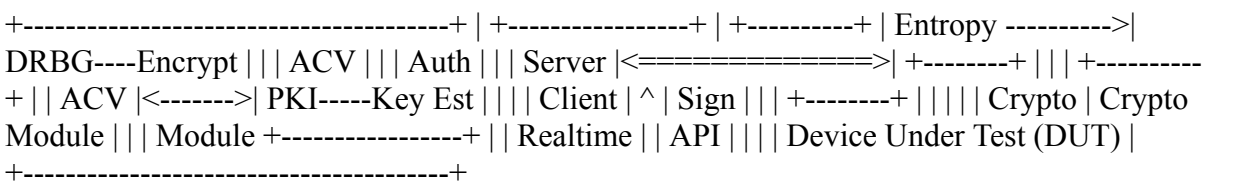


Figure 1

3.2. Server/Client Architecture(not realtime)

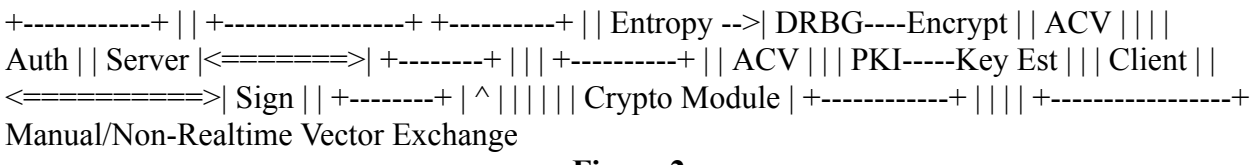


Figure 2

3.3. Server/Proxy Architecture

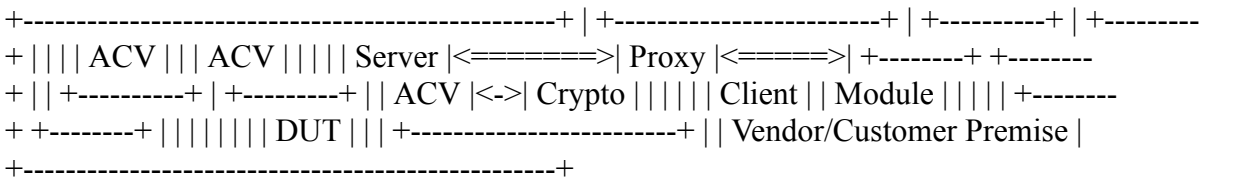


Figure 3

3.4. Terminology

The following terms are consistently used throughout this document and **SHOULD** be used throughout its extensions:

- Test Session — The largest structure of an instance of ACVP. Often generated from a single registration, it will contain many Test Vector Sets for the algorithms defined in the registration.
- Test Vector Set — The set of tests and data corresponding to an individual algorithm from within a registration. Uniquely identified throughout an instance of ACVP by the vsId. Contains many Test Groups.
- Test Group — The set of tests that share common properties within a Test Vector Set. Uniquely identified within the Test Vector Set by the tgId. Contains many Test Cases.
- Test Case — The smallest unit of the tests that represents an individual testable operation as defined by the parent Test Group. Uniquely identified throughout the Test Vector Set by the tcId.
- Registration — The JSON from the client to the server that describes the algorithms and capabilities for which the client is seeking test cases and a validation. A Registration can submit capabilities for multiple algorithms. Each algorithm will be broken into individual Test Vector Sets.
- Request — The JSON sent from the server to the client defining a single Test Vector Set to use as inputs for testing. Exactly one request will exist for each Test Vector Set.
- Response — The JSON from the client providing the desired cryptographic output for each of the Test Cases defined in the corresponding Request file. These are linked via the Test Vector Set vsId. As exactly one request exists for each vsId, one response will exist for each request.
- Disposition — The JSON from the server after the Response is submitted by the client detailing the correctness of the Test Cases. A “passed” disposition indicates that the particular algorithm in the Test Vector Set is ready for validation.
- OE — Operational Environment — The specific hardware and/or software the client’s cryptographic implementation uses to run.
- Realtime — For the purposes of this document realtime is defined as the client receiving a vector set and immediately performing the tests and returning the results to the server. In general, this is a case where the ACVP client and crypto module reside in the same box and directly communicate with each other. Non-realtime would refer to the case where the client may gather vector sets from the server and through some means introduce them to the crypto module gather the results and send them back to the server. In general, this is a case where the device hosting the crypto module cannot communicate directly with the ACVP client.

4. ACV Protocol

The ACV protocol will utilize existing mechanisms for transport coordinated with JSON formatted messaging.

+--+ | JSON Formatted ACVP request/ response messages | +--+ | HTTP[S] message transfer and signaling | +--+ | TLS for transport security(recommended) | +--+ | TCP for transport | | _____|

Figure 4 — Protocol Layering

4.1. HTTP URI Hierarchy

+------------+-----------------+------------------------+ server path prefix resource +------------ +-----------------+------------------------+ https://acvts.nist.gov/validation/acvp/v1/testSessions/1 +------------+----------+----+--+------------------------+ context API | version

Figure 5

Note that deployments utilizing ACV Proxy server **MAY** use a different protocol, e.g., HTTP, custom server, context and port number to interact with the DUT.

4.2. HTTP URI Resources

In the table below, any parts of a resource path enclosed in curly braces, { or }, are replaced by an instance of what is described in the braces. For example {testSessionId} could be replaced with 1.

An empty cell for a resource and HTTP Method combination denotes that the server returns an HTTP Status 405 code *Method not allowed (405)*.

Table 1 — Resources and their Available Operations

Resource	GET (read)	POST (create)	PUT (update)	DELETE
/requests	Section 10.7.1			
/requests/{requestId}	Section 10.7.2			
/vendors	Section 10.8.1	Section 10.8.2		
/vendors/{vendorId}	Section 10.8.3		Section 10.8.4	Section 10.8.5
/vendors/{vendorId}/addresses	Section 10.9.1			
/vendors/{vendorId}/addresses/{addressId}	Section 10.9.2			

Resource	GET (read)	POST (create)	PUT (update)	DELETE
/vendors/{vendorId}/contacts	Section 10.8.6			
/persons	Section 10.10.1	Section 10.10.2		
/persons/{personId}	Section 10.10.3		Section 10.10.4	Section 10.10.5
/oes	Section 10.12.1	Section 10.12.2		
/oes/{oeId}	Section 10.12.3		Section 10.12.4	Section 10.12.5
/modules	Section 10.11.1	Section 10.11.2		
/modules/{moduleId}	Section 10.11.3		Section 10.11.4	Section 10.11.5
/dependencies	Section 10.13.1	Section 10.13.2		
/dependencies/properties	Section 10.13.3			
/dependencies/{dependencyId}	Section 10.13.4		Section 10.13.5	Section 10.13.6
/algorithms	Section 10.14.1			
/algorithms/{algorithmId}	Section 10.14.2 (Optional)			
/testSessions	Section 10.16.1 (Optional)	Section 10.16.2		
/testSessions/{testSessionId}	Section 10.16.3 (Optional)		Section 10.16.4	Section 10.16.5
/testSessions/{testSessionId}/results	Section 10.16.6			
/testSessions/{testSessionId}/vectorSets	Section 10.17.1			
/testSessions/{testSessionId}/vectorSets/{vectorSetId}	Section 10.17.2			Section 10.17.3
/testSessions/{testSessionId}/vectorSets/{vectorSetId}/results	Section 10.17.4	Section 10.17.5	Section 10.17.6	
/testSessions/{testSessionId}/vectorSets/{vectorSetId}/expected	Section 10.17.7 (Optional)			

The resource path is appended to the path prefix to form the URI used with an HTTP Method to perform the desired ACVP operation. For example to create a new test session using the "/testSessions" resource is "/acvp/v1/testSessions" (assuming an empty context). To create a new Test Session, the ACVP client would use the following HTTP request-line:

POST /acvp/v1/testSessions HTTP/1.1

Likewise, to request a specific vector set from the server the ACVP client would use the following request-line:

GET /acvp/v1/testSessions/1/vectorSets/1 HTTP/1.1

5. Security Considerations

It is **RECOMMENDED** that HTTPS and TLS 1.2 or greater be used in order to enforce a secure communication method. Not all environments will have TLS so HTTP with some level of authentication may be the only option.

5.1. Authentication

It is **RECOMMENDED** that an authentication scheme be used. Typically, a JSON Web Token (JWT) is created by the server upon successful client authentication and returned to the client to use as an authorization mechanism for accessing the server resources—see Section 10.3 below for more information. Depending on the target environment and usage objectives, the authentication can be as weak as basic HTTP authentication or as strong as TLS mutual certificate authentication. Definition of an authentication scheme will not be discussed here, but should be agreed upon between the client and server owning entities including the servers owned by the validation authorities. The Section 8 endpoint will be discussed for the purposes of establishing a test session but how a client is authenticated to the server is not prescribed.

6. Encoding

The encoding used for the request/response messaging will be JSON ([RFC 7159]). The data will be identified by: Content-type: application/json In order to allow environment specific extensions to a particular version of the ACV protocol, a top-level JSON keyword, extensions will be used to extend the OE description and/or the capabilities. Extensions **MAY** be ignored by the ACV server. Vector and vector response data will be JSON encoded.

7. Submission Size Considerations

Some server implementations **MAY** require alternative handling for submission sizes that exceed resource limitations of the normal workflow for a `POST` response. Server implementations **MUST** indicate whether the server requires large submission handling and what the maxmimum value for non-large submissions is within the Section 8 response.

8. Login

A login endpoint is mentioned throughout the document and although how to authenticate is left up to the server the following description is an example of the login endpoint to establish a session and obtain a Section 10.3.1. See Section 11 for more information on how to handle data which exceeds the `sizeConstraint` in the event the value of `largeEndpointRequired` is `true`.

- **password** — `string`, a property for providing a password value
- **accessToken** — `string`, a JWT associated with the current user for which renewal of the expiration is desired
- **largeEndpointRequired** — `boolean`, true if the server requires the large endpoint for submissions with a size that exceeds the value found in **sizeConstraint**.
- **sizeConstraint** — `number`, provides the maximum value, in bytes, of a submission not required to use the Section 11 or -1 if `largeEndpointRequired` is `false`.

8.1. Request

There are two forms of a login request. There is the initial form which just provides the authentication information without any JWT, and there is a renewal login that allows a user to obtain a new JWT containing the claims from an expired JWT in order to access a resource protected with those claims.

POST /login (Initial)

```
[
  {"acvVersion": "{acvp-version}"},
  {
    "password": "{password}"
  }
]
```

Figure 6

POST /login (Renewal)

```
[
  {"acvVersion": "{acvp-version}"},
  {
    "password": "{password}",
    "accessToken": "{jwt value}"
  }
]
```

Figure 7

8.2. Response

```
[
  {"acvVersion": "{acvp-version}"},
  {
    "accessToken": "{jwt value}",
    "largeEndpointRequired": false,
    "sizeConstraint": -1
  }
]
```

Figure 8

8.3. Multi Refresh JWT

This endpoint can be utilized for refreshing multiple JWTs with a single POST. Because a JWT with claims is issued for each test session created and clients have the option of performing more than one test session at a time, a mechanism for refreshing multiple JWTs across multiple test sessions simultaneously **MAY** be made available.

8.3.1. Request

POST /login/refresh

```
[
  {"acvVersion": "{acvp-version}"},
  {
    "password": "{password}",
    "accessToken": [
      "{jwt1}",
      "{jwt2}",
      "..."
    ]
  }
]
```

Figure 9

8.3.2. Response

```
[
  {"acvVersion": "{acvp-version}"},
  {
    "accessToken": [
      "{newJwt1}",
      "{newJwt2}",
      "..."
    ],
    "largeEndpointRequired": false,
```

```
    "sizeConstraint": -1
  }
]
```

Figure 10

Note the order of JWTs between the request and response is preserved.

9. Versioning

The version of the ACVP protocol will be carried with each message and will contain a simple major.minor format. Major version changes will not be backward compatible, however additions and enhancements that do not disrupt compatibility will be indicated with a minor version change. A server **MAY** accept a down-level version from the client if it can process at a lower level. If not, it will reject the session. All subsequent messages will carry the negotiated version value.

10. Messaging and Workflow

10.1. Product Registration/Capabilities Exchange

The product registration will utilize the URI resources Section 4.2 to register and provide cryptographic capabilities. This exchange will consist of several message exchanges and will provide a detailed list of cryptographic algorithms and their options to be tested during the testSession, see Figure 17, retrieval of test vectors, submission of test responses, determining test result, and a certification step to associate the product and tested environment information with the completed test. A set URI resources are also available to retrieve and manage the available metadata such as companay name, primary contact, product descriptions, and operational environment descriptions.

10.2. Test Exchange

The test exchange consists of the ACV client requesting a particular vectorSet associated with the testSession. The server responds with the vectorSet. The client **MAY** retreive and process the vectorSets in order and **MAY** retreive a vectorSet and immediately return results or request all of the vectorSets and return results at a later time. The client repeats this process until all of the vectorSets in the testSession list have been processed. Once a vectorSet result has been POSTed to the server the client may request success/failure results from the server at any time, however if vectorSets have not been completed the overall status will be incomplete. A minimal message flow is described below Figure 17.

10.3. JSON Web Token (JWT)

JSON Web Token is described in [RFC 7519] and is used as an authorization mechanism for gaining access to different resources.

```
{
    "alg" : "none"
}
{
    "iss" : "nist.gov",
    "nbf": 1598293915,
    "exp": 1598295715,
    "iat": 1598293915,
    "pkey" : "cc74f56acdba635079383a03941d68db55c7b3c2f (truncated)"
}
{

}
```

Figure 11 — Unsigned JWT

The JWT can be secured if desired using the header encryption “alg” value defined to HS256(HMAC-SHA256) or one of the other secure values. Key agreement would follow RFC7518.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
{
  "iss" : "nist.gov",
  "nbf": 1598293915,
  "exp": 1598295715,
  "iat": 1598293915,
  "pkey" : "cc74f56acdba635079383a03941d68db55c7b3c2f (truncated)"
}
{
    "{signature}"
}
```

Figure 12 — Signed JWT

where “{signature}” is made up of:

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

Figure 13

and where “HMACSHA256” is the algorithm specified in the JWT header.

The first four claims are required, however “pkey” is an optional private claim used to pass the key used for encrypting the database at the server. Enabling this option is discussed further in Section 10.16.2

10.3.1. Authorization flows with JWT

JSON Web Token is described in [RFC 7519] and is used as an authorization mechanism for gaining access to different resources.

In order to access any resource which requires authorization a client must supply the JWT as an `Authorization` header value as a `Bearer` token. An example header value is:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5c (truncated)"

Figure 14

Workflow authorization flows. All exchanges shown are over HTTP.

+--------+---------------------------------+------+--------+ | Client | |Server| Notes | +--------
+---------------------------------+------+--------+ | |POST to /login or similar with | | | | |appropriate

Figure 15

10.3.2. JWT Expiration/Renewal

The JWT access tokens received from either the /login server endpoint **SHALL** be set to expire after a pre-defined period. The specific length of the expiration period is out of scope for this specification. However, the expiration period length impacts both the security and protocol overhead. Longer expiration periods reduce the overhead but increase the window for attacks. Attempting to access a service with an expired JWT **SHALL** result in a "401 Unauthorized" HTTP status code.

A client may renew an expired JWT access token using the mechanism shown in Figure 16below.

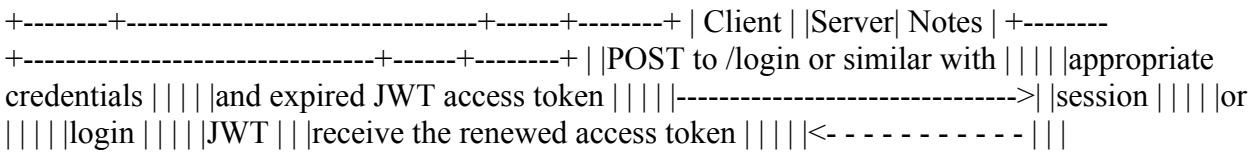


Figure 16 — JWT access token renewal flows. All exchanges shown are over HTTP.

10.4. Message Flow

An example minimum message flow between client and server after receiving the JWT is seen in the figure below.

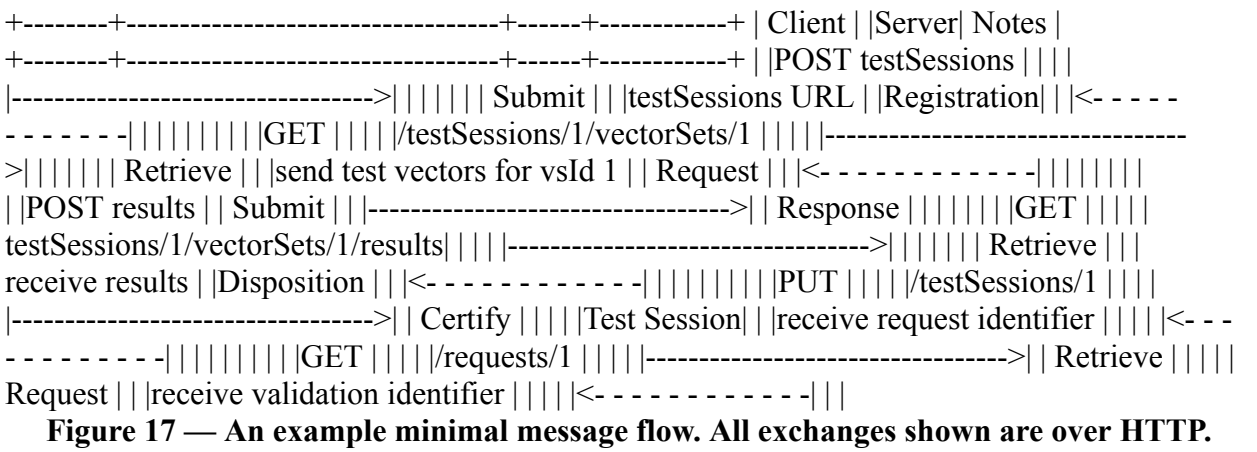
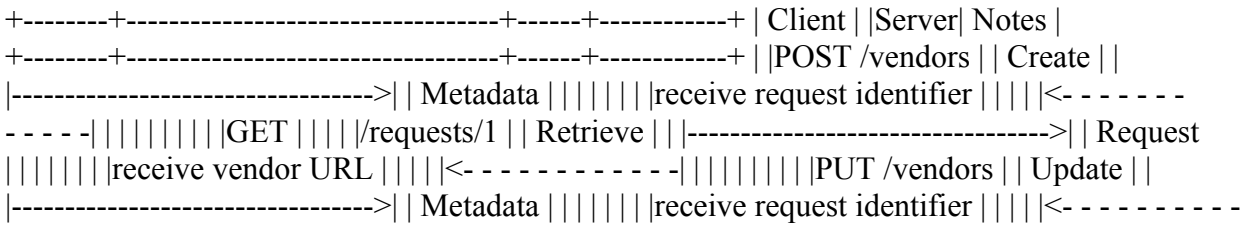


Figure 17 — An example minimal message flow. All exchanges shown are over HTTP.

Metadata creation and update example. The list of available metadata endpoints can be found in Section 4.2.



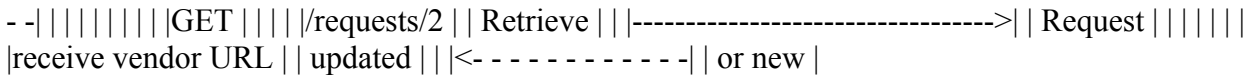


Figure 18

In the event a submission response exceeds server defined thresholds the following workflow will need to be followed in order to submit the test result. See Section 11 for more information.

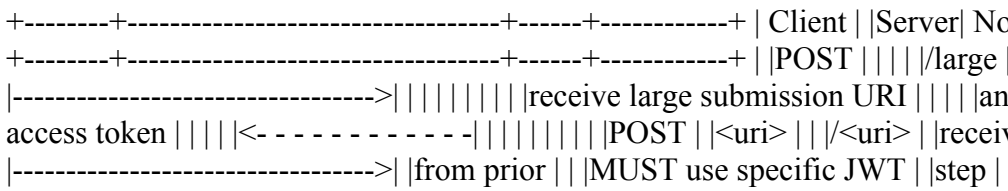


Figure 19

10.5. Paging

Some resource operations require paging in order to avoid returning large amounts of data. Each operation that uses paging will indicate that uses paging and what the value for each element will be within the section describing that operation. All paged responses **MUST** follow the format described in Section 10.5.2. Conversely, clients may navigate pages using the paging parameters described in Section 10.5.1. Server implementations **SHOULD** impose limitations on the page size limit based on resource constraints.

10.5.1. Parameters

A Server **MUST** accept requests without paging parameters. If not all results are returned, the response **MUST**indicate that not all of the results were provided using the `incomplete` property of a paged response described in Section 10.5.2. The query parameters clients **MUST** use to specify paging are described below:

- **limit**—`number`, the maximum number of entries to return. Server implementations **MUST** allow requests without a provided value, but the default value is a choice for server implementations.
- **offset**—`number`, the offset into the list of entries, **MUST** default to `0` if not provided.

GET /acvp/v1/vendors?offset=20&limit=20 HTTP/1.1

10.5.2. Response

A paged response has the following properties:

- **totalCount**—`number`, the total number of resources available to return
- **incomplete**—`boolean`, true if more resources are avaiable than what is returned in the response

- **links** — `object`, links to use when navigating the pages
  - **first** — `string`, a link to the first page in the result set
  - **next** — `string`, a link to the next page in the result set, `null` if no next page is available
  - **prev** — `string`, a link to the previous page in the result set, `null` if no previous page is available
  - **last** — `string`, a link to the last page in the result set
- **data** — `array`, contains an array of data appropriate to the resource requested

[{"acvVersion": <acvp-version>}, { "totalCount" : 22007, "incomplete" : true, "links" : { "first" : "/acvp/v1/<resource>?offset=0&limit=20", "next" : "/acvp/v1/<resource>?offset=20&limit=20", "prev" : null, "last" : "/acvp/v1/<resource>?offset=22000&limit=20" }, "data" : [<resource response>] }]

Figure 20

10.6. Query Parameters

Some of the resource listing operations allow for query parameters to be provided to filter out the returned values. Each resource will list what properties and operations are available but the general format of the query parameter string is consistent across all resources. The format allows for the specification of complex filters with the concept of groups, where all elements in the same group **MUST** be AND'ed together and different groups are OR'ed together. The URL including the parameter values **MUST** conform to [RFC 3986] and **MUST** use UTF-8 character encoding.

General format of a query parameter element.

<property>[<index>]=<operation>:<value>

Figure 21

- **property** — is the property to be specified
- **index** — is an arbitrary group index, elements with same group index are AND'ed together and elements with different indices are OR'ed togther. `index` **MUST** be between 0 and 99 inclusive.

- **operation** — is an operation on a property and a value. Not all operations require a value, and not all properties will permit every operation. Available operations are:
  - **eq** — filter based on property equal to the value
  - **ne** — filter based on the property not equal to the value
  - **gt** — filter based on the property greater than the value
  - **ge** — filter based on the property greater than or equal to the value
  - **lt** — filter based on the property less than the value
  - **le** — filter based on the property less than or equal to the value
  - **contains** — filter based on the property containing the value
  - **start** — filter based on the property starting with the value
  - **end** — filter based on the property ending with the value
- **value** — the value to filter on, it **MAY** be constrained based on the property

Example 1

/resource?property1[0]=eq:foo&property2[0]=eq:foo
&property1[1]=eq:test&property2[1]=ne:bar

Figure 22

For the example above the results returned would include resources that have: `property1` equal to `foo` and `property2` equal to `foo` or resources that have `property1` equal to `test` and `property2` not equal to `bar`.

Example 2 based on Section 10.8.1.

/vendors?name[0]=contains:acme&name[1]=contains:test

Figure 23

For the example above the vendor results returned would include resources that have a `name` property value that contains either `acme` or `test`.

10.7. Requests

Some resource operations make a request to modify or create data. To facilitate an out-of-band approval step, where data can be inspected to insure it meets the business requirements of the validation authority which operates the server, the operations will return a `request` url that can be used to obtain information about the status and disposition of the requested modification. Whether or how an authority implements an approval step is outside the scope of this specification.

A request resource is not externally updateable, but **SHOULD** update based on server processing. The properties for a request response:

- **url** — `string`, identifier for this resource

- **status** — `string`, one of:
  - **initial** — initial state of the request, created
  - **processing** — server is processing the request
  - **approved** — the requested operation was successfully processed
  - **rejected** — the requested operation was rejected and no change was made
- **message** — `string`, a placeholder for any message describing a rejection
- **approvedUrl** — `string`, a link to the resource which was created or modified as a result of the requested operation

10.7.1. Request Listing

GET /requests

Returns a paged listing of requests for the current user. Each element in the `data` array is a `request object` as described in Section 10.7.2. See also Section 10.5.2 for a description of a paged response.

10.7.2. Request Information

GET /requests/{requestId}

Retrieve Information for a specific request

10.7.2.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "url": "/acvp/v1/requests/2",
        "status": "approved",
        "approvedUrl" : "/acvp/v1/vendors/2"
    }
]
```

Figure 24

10.8. Vendor Resources

The available properties for vendor resources are:

- **url** — `string`, identifier for the vendor resource within which this property is located
- **name** — `string`
- **parentUrl** — a parent vendor identifier, allows for multiple divisions or business units to share a parent company identifier
- **website** — `string`

- **emails** — array of `string`
- **phoneNumbers** — array of phone objects,
  - **number** — `string`
  - **type** — `string`, one of (fax, voice)
- **contactsUrl** — `string`, identifier for the list of person resources associated with this vendor
- **addresses** — an address object,
  - **url** — `string`, identifier for the address resource
  - **street1** — `string`
  - **street2** — `string`
  - **street3** — `string`
  - **locality** — `string`
  - **region** — `string`
  - **country** — `string`
  - **postalCode** — `string`

10.8.1. Vendor Listing

GET /vendors

Returns a paged listing of vendors. Each element in the `data` array is a `vendor object` as described in Section 10.8.3. See also Section 10.5.2 for a description of a paged response.

Available Section 10.6:

- **name**: `eq`, `start`, `end`, `contains`
- **website**: `eq`, `start`, `end`, `contains`
- **email**: `eq`, `start`, `end`, `contains`
- **phoneNumber**: `eq`, `start`, `end`, `contains`

10.8.2. Create a New Vendor

POST /vendors

Request the creation of a new Vendor.

10.8.2.1. Request

`name` is required and all other defined properties are OPTIONAL. Any additional properties included in the request are ignored.

```
[
    {"acvVersion": "{acvp-version}"},
    {
```

```
      "name": "Acme, LLC",
      "website": "www.acme.acme",
      "emails" : [ "inquiry@acme.acme" ],
      "phoneNumbers" : [{
          "number" : "555-555-1234",
          "type" : "voice"
      }],
      "addresses" : [{
          "street1" : "123 Main Street",
          "locality" : "Any Town",
          "region" : "AnyState",
          "country" : "USA",
          "postalCode" : "123456"
      }]
    }
]
```

Figure 25

10.8.2.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the vendor resource which was created. The url of any resources created incidental to the creation of the vendor resource would be available through the Section 10.8.3 operation.

Reply is a request response as described in Section 10.7.

10.8.3. Vendor Information

GET /vendors/{vendorId}

Retrieve Information for a specific vendor

10.8.3.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "url": "/acvp/v1/vendors/2",
        "name": "Acme, LLC",
        "website": "www.acme.acme",
        "emails" : [ "inquiry@acme.acme" ],
        "phoneNumbers" : [{
          "number" : "555-555-1234",
          "type" : "voice"
        }],
        "contactsUrl": "/acvp/v1/vendors/2/contacts",
        "addresses" : [{
```

```
            "url" : "/acvp/v1/vendors/1/addresses/4",
            "street1" : "123 Main Street",
            "locality" : "Any Town",
            "region" : "AnyState",
            "country" : "USA",
            "postalCode" : "123456"
        }]
    }
]
```

Figure 26

10.8.4. Update an existing Vendor

PUT /vendors/{vendorId}

Update a vendor

The `url` property is not updateable.

10.8.4.1. Request

Can be any subset of the updateable properties. If a property is not included its value is not changed. A `null` value for a property indicates the value should be removed.

When updating the addresses array, the `url` of every address resource to be kept **MUST** be included. Any missing addresses will be removed and any new addresses will be created.

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "name": "Acme, LLC",
        "website": "www.acme.acme",
        "emails" : [ "inquiry@acme.acme" ],
        "addresses" : [{
                "url" : "/acvp/v1/vendors/2/addresses/4",
                "street1" : "123 Main Street",
                "locality" : "Any Town",
                "region" : "AnyState",
                "country" : "USA",
                "postalCode" : "123456"
        }]
    }
]
```

Figure 27

10.8.4.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the vendor resource which was updated. A server implementation **MAY** create a new resource instead of updating the existing resource.

10.8.5. Remove a Vendor

DELETE /vendors/{vendorId}

Request to delete a specific vendor. Reply is a request response as described in Section 10.7.

The server is not required to remove the resource but **MUST** return a `rejection`value for the `status` property if the resource will not be removed.

10.8.6. Contact Listing for a Vendor

GET /vendors/{vendorId}/contacts

Returns a paged listing of persons specific to the vendor. Each element in the `data` array is a `person object` as described in Section 10.10.3. See also Section 10.5.2 for a description of a paged response.

10.9. Address Resources

The available properties for address resources are:

- **url** — `string`, identifier for this resource
- **street1** — `string`
- **street2** — `string`
- **street3** — `string`
- **locality** — `string`
- **region** — `string`
- **country** — `string`
- **postalCode** — `string`

10.9.1. Address Listing

GET /vendors/{vendorId}/addresses

Returns a paged listing of addresses for the vendor. Each element in the `data` array is an `address object` as described in Section 10.9.2. See also Section 10.5.2 for a description of a paged response.

The addresses returned are equivalent to the address array returned in Section 10.8.3 for the same vendor resource.

10.9.2. Address Information

GET /vendors/{vendorId}/addresses/{addressId}

Retrieve Information for a specific address

10.9.2.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "url" : "/vendors/2/addresses/4",
        "street1" : "123 Main Street",
        "locality" : "Any Town",
        "region" : "AnyState",
        "country" : "USA",
        "postalCode" : "123456"
    }
]
```

Figure 28

10.10. Person Resources

The available properties for person resources are:

- **url** — `string`, identifier for this resource
- **fullName** — `string`
- **vendorUrl** — `string`, identifier for the vendor resource this person is associated with
- **emails** — array of `string`
- **phoneNumbers** — array of phone objects,
  - **number** — `string`
  - **type** — `string`, one of (fax, voice)

The email and phone number values are specific to the person resource and are independent of the equivalent information in the vendor resource.

10.10.1. Person Listing

GET /persons

Returns a paged listing of persons. Each element in the `data` array is a `person object` as described in Section 10.10.3. See also Section 10.5.2 for a description of a paged response.

Available Section 10.6:

- **fullName**: `eq`, `start`, `end`, `contains`

- **email**: `eq`, `start`, `end`, `contains`
- **phoneNumber**: `eq`, `start`, `end`, `contains`
- **vendorId**: `eq`, `ne`, `lt`, `le`, `gt`, `ge`

10.10.2. Create a New Person

POST /persons

Request the creation of a new Person.

10.10.2.1. Request

`fullName` and `vendorUrl` are required. Other defined resource properties are OPTIONAL. Any additional properties included in the request are ignored.

```
[
    {"acvVersion": "{acvp-version}"},
    {
      "fullName": "Jane Smith",
      "vendorUrl" : "/acvp/v1/vendors/2",
      "emails": ["jane.smith@acme.acme"],
      "phoneNumbers" : [
          {
              "number": "555-555-0001",
              "type" : "fax"
          }, {
              "number": "555-555-0002",
              "type" : "voice"
          }
      ]
    }
]
```

Figure 29

10.10.2.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the person resource which was created.

10.10.3. Person Information

GET /persons/{personId}

Retrieve Information for a specific person

10.10.3.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
```

```
    {
      "url": "/acvp/v1/persons/4",
      "fullName": "Jane Smith",
      "vendorUrl" : "/acvp/v1/vendors/2"
      "emails": ["jane.smith@acme.acme"],
      "phoneNumbers" : [
          {
              "number": "555-555-0001",
              "type" : "fax"
          }, {
              "number": "555-555-0002",
              "type" : "voice"
          }
      ]
    }
]
```

Figure 30

10.10.4. Update an existing Person

PUT /persons/{personId}

Update a person

The `url` property is not updateable.

10.10.4.1. Request

Can be any subset of the updateable properties. If a property is not included its value is not changed. A `null` value for a property indicates the value should be removed.

```
[
    {"acvVersion": "{acvp-version}"},
    {
      "fullName": "Jane Smith",
      "emails": ["jane.smith@acme.acme"],
      "phoneNumbers" : [
          {
              "number": "555-555-0001",
              "type" : "fax"
          }, {
              "number": "555-555-0002",
              "type" : "voice"
          }
      ]
    }
```

```
]
```

Figure 31

10.10.4.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the person resource which was updated. A server implementation **MAY** create a new resource instead of updating the existing resource.

10.10.5. Remove a Person

DELETE /persons/{personId}

Request to delete a specific person. Reply is a request response as described in Section 10.7.

The server is not required to remove the resource but **MUST** return a `rejection` value for the `status` property if the resource will not be removed.

10.11. Modules

The available properties for module resources are:

- **url** — `string`, identifier for this resource
- **name** — `string`
- **version** — `string`
- **type** — `string`, valid values are:
  - `Software` — software-based modules
  - `Hardware` — hardware-based modules
  - `Firmware` — firmware-based modules
- **website** — `string`
- **vendorUrl** — `string`, identifier for a Section 10.8
- **addressUrl** — `string`, identifier for an Section 10.9
- **contactUrls** — `string array`, array of identifiers for a Section 10.10
- **description** — `string`, a description of the implementation

10.11.1. List Modules

GET /modules

Returns a paged listing of modules. Each element in the `data` array is a `module object` as described in Section 10.11.3. See also Section 10.5.2 for a description of a paged response.

Available Section 10.6:

- **name**: `eq`, `start`, `end`, `contains`
- **version**: `eq`, `start`, `end`, `contains`
- **website**: `eq`, `start`, `end`, `contains`
- **type**: `eq`, `ne`
- **vendorId**: `eq`, `ne`, `lt`, `le`, `gt`, `ge`
- **description**: `eq`, `start`, `end`, `contains`

10.11.2. Register a new Module

POST /modules

Register a new module.

10.11.2.1. Request

`name`, `vendorUrl`, and `description` are required. Other defined resource properties are OPTIONAL. Any additional properties included in the request are ignored.

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "name": "ACME ACV Test Module",
        "version": "3.0",
        "type": "Software",
        "vendorUrl": "/acvp/v1/vendors/2",
        "addressUrl": "/acvp/v1/vendors/2/addresses/4",
        "contactUrls": ["/acvp/v1/persons/1" ],
        "description" : "ACME module with more"
    }
]
```

Figure 32

10.11.2.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the module resource which was created. The url of any resources created incidental to the creation of the module resource would be available through the Section 10.11.3 operation.

10.11.3. Retrieve information for a Module

GET /modules/{moduleId}

Returns information about a specific module.

10.11.3.1. Response

```
[
```

```
    {"acvVersion": "{acvp-version}"},
    {
        "url": "/acvp/v1/modules/2",
        "name": "ACME ACV Test Module",
        "version": "2.0",
        "type": "Software",
        "website" : "www.acme.acme",
        "vendorUrl": "/acvp/v1/vendors/2",
        "addressUrl": "/acvp/v1/vendors/2/addresses/4",
        "contactUrls": ["/acvp/v1/persons/1" ],
        "description": "ACME module with features."
    }
]
```

Figure 33

10.11.4. Update a Module

PUT /modules/{moduleId}

Update an existing module.

It may not be possible to update all properties of a module once the module has been associated with a test session.

10.11.4.1. Request

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "description" : "ACME module with more"
    }
]
```

Figure 34

10.11.4.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the module resource which was updated. A server implementation **MAY** create a new resource instead of updating the existing resource.

10.11.5. Delete a Module

DELETE /modules/{moduleId}

Request to delete a specific module. Reply is a request response as described in Section 10.7.

The server is not required to remove the resource but **MUST** return a `rejection` value for the `status` property if the resource will not be removed.

10.12. Operational Environments (OEs)

The available properties for operational environment resources are:

- **url** — `string`, identifier for this resource
- **name** — `string`
- **dependencyUrls** — an array of `string` which identify the Section 10.13 which comprise this OE.
- **dependencies** — an array of Section 10.13s which comprise this OE. Only valid on update or create and **MAY** be used in combination with the `dependencyUrls` property.

10.12.1. List Operational Environments

GET /oes

Returns a paged listing of available operational environments. Each element in the `data` array is a `operational environment object` as described in Section 10.12.3. See also Section 10.5.2 for a description of a paged response.

Available Section 10.6:

- **name**: `eq`, `start`, `end`, `contains`

10.12.2. Create a new Operational Environment

POST /oes

Create a new operational environment.

10.12.2.1. Request

`name` is required. Other defined resource properties are OPTIONAL. Any additional properties included in the request are ignored.

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "name": "Ubuntu Linux 3.1 on AMD 6272 Opteron Processor
                 with Acme installed",
        "dependencyUrls": [
            "/acvp/v1/dependencies/4",
            "/acvp/v1/dependencies/5",
            "/acvp/v1/dependencies/7"
        ]
    }
]
```

Figure 35

10.12.2.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the operational environment resource which was created. The url of any resources created incidental to the creation of the operational environment resource would be available through the Section 10.8.3 operation.

10.12.3. Retrieve information for an Operational Environment

GET /oes/{oeId}

Returns information about a specific operational environment.

10.12.3.1. Response


```
[
  {
    "acvVersion": "{acvp-version}"
  },
  {
    "url": "/acvp/v1/oes/21495",
    "name": "Test DMC0428 Inline Ubuntu Linux 3.1 on AMD 6272 Opteron
 Processor with Acme package installed",
    "dependencies": [
      {
        "url": "/acvp/v1/dependencies/23563",
        "type": "software",
        "name": "Linux 3.1 DMC0427 Extra",
        "description": "Testing0427 cpe-2.3:o:ubuntu:linux:04.27"
      },
      {
        "url": "/acvp/v1/dependencies/23564",
        "type": "software",
        "name": "Linux 4.3 DMC0428 A1",
        "description": "Testing0428 A1 cpe-2.3:o:ubuntu:linux:4.3",
        "cpe": "cpe-2.3:oa1:ubuntu:linux:4.3"
      },
      {
        "url": "/acvp/v1/dependencies/23565",
        "type": "software",
        "name": "Linux 4.3 DMC0428 B1",
        "description": "Testing0428 B1 cpe-2.3:o:ubuntu:linux:4.3",
        "cpe": "cpe-2.3:ob1:ubuntu:linux:4.3"
      },
      {
        "url": "/acvp/v1/dependencies/23566",
        "type": "software",
        "name": "Linux 4.3 DMC0428 C1",
```

```
        "description": "Testing0428 C1 cpe-2.3:o:ubuntu:linux:4.3",
        "cpe": "cpe-2.3:oc1:ubuntu:linux:4.3"
      }
    ]
  }
]
```

Figure 36

10.12.4. Update an Operational Environment

PUT /oes/{oeId}

Update an existing operational environment.

It may not be possible to update all (or any) properties of an operational environment resource once the resource has been associated with a test session.

10.12.4.1. Request

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "name": "Windows 10 on Intel Xeon 5100 Series Processor",
    }
]
```

Figure 37

10.12.4.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the operational environment resource which was updated. A server implementation **MAY** create a new resource instead of updating the existing resource.

10.12.5. Delete an Operational Environment

DELETE /oes/{oeId}

Request to delete an operation environment. Reply is a request response as described in Section 10.7.

The server is not required to remove the resource but **MUST** return a `rejection` value for the `status` property if the resource will not be removed.

10.13. Dependencies

An operational environment is composed of one or more dependencies which fully characterize and describe the operational environment under which a module was tested. An operational environment **MAY** have many different types of dependencies.

The available properties for dependency resources are:

- **url** — `string`, identifier for this resource
- **type** — `string`, the type of the dependency, a non-inclusive list of values that **MAY** be allowed are:
  - **os** — operating system
  - **cpu** — Central Processing Unit (CPU) chip
  - **software** — a software dependency
  - **firmware** — a firmware dependency
- **name** — `string`, a short name of the dependency
- **description** — `string`, a longer description of the dependency providing any additional detail that may be useful
- `{varies}` the value of `type` for a dependency **MAY** require or allow for different name/value pairs to be added to a dependency to better describe and define the dependency which in turn describes the operational environment that a module will operate under. The possible name/value pairs for a given value of `type` **MAY** be provided by the response of Section 10.13.3, if the server implements this endpoint. Otherwise a server **MAY** choose to restrict or not restrict the range of name/value pairs available, but any restrictions **MUST** be clearly documented.

10.13.1. List Dependencies

GET /dependencies

Returns a paged listing of available dependencies. Each element in the `data` array is a `dependency object` as described in Section 10.13.4. See also Section 10.5.2 for a description of a paged response.

Available Section 10.6:

- **name**: `eq`, `start`, `end`, `contains`
- **type**: `eq`, `start`, `end`, `contains`
- **description**: `eq`, `start`, `end`, `contains`

10.13.2. Register a new Dependency

POST /dependencies

Register a new dependency.

10.13.2.1. Request

`name` is required. Other defined resource properties are OPTIONAL. Any additional properties included in the request are ignored.

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "type": "software",
        "name": "Linux 3.1",
        "description" : "Ubuntu Linux Distribution 3.1",
        "cpe": "cpe-2.3:o:ubuntu:linux:3.1"
    }
]
```

Figure 38

10.13.2.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the dependency resource which was created.

10.13.3. List Dependency Properties

(Optional) **GET /dependencies/properties**

Returns a Section 10.5 list of available dependency properties.

An array of property objects is returned with the following properties:

- **name** — `string`
- **dataType** — `string`
- **validTypes** — an array of `string` where each element corresponds to a dependency type value that this property may be used with.
- **description** — `string`

10.13.3.1. Example Dependency Property Elements

```
    {
       "name": "swid",
       "dataType": "string",
       "validTypes": ["software"],
       "description": "A Software identification (SWID) tag as
        described in ISO/IEC 19770-2:2015. NIST IR 8060,
        https://csrc.nist.gov/publications/detail/nistir/8060/final,
        provides guidance on creating and maintaining SWID tags."
    },
```

```
    {
       "name": "cpe",
       "dataType": "string",
       "validTypes": [
           "software",
           "processor"
       ],
       "description": "A Common Platform Enumeration (CPE)
        formatted name according to Version 2.3 of the CPE
        Naming Specification found in NISTIR 7695,
        https://csrc.nist.gov/publications/detail/nistir/7695/final."
    },
    {
       "name": "manufacturer",
       "dataType": "string",
       "validTypes": ["processor"],
       "description": "The name of the manufacturer of
                       the processor dependency."
    },
    {
       "name": "family",
       "dataType": "string",
       "validTypes": ["processor"],
       "description": "The name of the family of the processor."
    },
    {
       "name": "series",
       "dataType": "string",
       "validTypes": ["processor"],
       "description": "The name of the series of the processor."
    }
```

Figure 39

10.13.4. Retrieve information for a Dependency

GET /dependencies/{dependencyId}

Returns information about a specific dependency.

10.13.4.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "type": "software",
        "name": "Linux 3.1",
        "description" : "Ubuntu Linux Distribution 3.1",
```

```
        "cpe": "cpe-2.3:o:ubuntu:linux:3.1"
    }
]
```

Figure 40

10.13.5. Update a Dependency

PUT /dependencies/{dependencyId}

Update an existing dependency.

It may not be possible to update all (or any) properties of a dependency resource once the resource has been associated with an operational environment.

10.13.5.1. Request

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "name": "Linux 3.1.0",
    }
]
```

Figure 41

10.13.5.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the dependency resource which was updated. A server implementation **MAY** create a new resource instead of updating the existing resource.

10.13.6. Delete a Dependency

DELETE /dependencies/{dependencyId}

Request to delete a dependency. Reply is a request response as described in Section 10.7.

The server is not required to remove the resource but **MUST** return a `rejection` value for the `status` property if the resource will not be removed.

10.14. Algorithms

The Algorithm resources are informational only.

10.14.1. Algorithms Listing

GET /algorithms

Returns a list of available algorithms on the server.

10.14.1.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
    {"algorithms": [
        {
            "url": "/acvp/v1/algorithms/2",
            "name": "AES",
            "mode": "GCM",
            "versions": [
                "{acvp-version1}",
                "{acvp-version2}"
            ]
        },
        {
            "url": "/acvp/v1/algorithms/3",
            "name": "AES",
            "mode": "ECB",
            "versions": [
                "{acvp-version}"
            ]
        }
    ]}
]
```

Figure 42

10.14.2. Algorithm Information

GET /algorithms/{algorithmId}

Retrieve Information for about a specific algorithm.

10.14.2.1. Response

Response may vary from server depending on internal representation.

10.15. Validations

The Validations resources are informational only.

10.15.1. Validation Information

GET /validations/{validationId}

Retrieve information about a specific validation.

10.15.1.1. Response

Response **MAY** vary from server depending on internal representation. Available properties for validations **MAY** include (but are not limited to):

- **url** — `string`, identifier for this resource
- **validationId** — `string`, unique representation of the validation and source.
- **moduleUrl** — `string`, the module URL associated with this validation. See Section 10.11
- **oeUrls** — `array of string`, the Operational Environments associated with this validation. See Section 10.12

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "url": "/acvp/v1/validations/50",
        "validationId": "A12",
        "moduleUrl": "/acvp/v1/modules/1",
        "oeUrls": [
            "/acvp/v1/oes/1"
        ]
    }
]
```

Figure 43

10.16. Test Sessions

The available properties for test session resources are:

- **url** — `string`, identifier for this resource
- **acvpVersion** — `string`, version of ACV protocol used to created the test session.
- **createdOn** — Section 15.1
- **expiresOn** — Section 15.1
- **encryptAtRest** — `boolean`
- **vectorSetsUrl** — `string`, resource for all of the vector sets
- **publishable** — `boolean`, indicates whether this test session may be submitted for validation
- **passed** — `boolean`, indicates whether all of the vector set tests have passed
- **isSample** — `boolean`, if true Section 10.17.7 will return expected result values. As well, Test Vector Sets **MAY** contain fewer Test Cases for quicker generation and verification.

10.16.1. Test Session Listing (Current User)

GET /testSessions

This is an OPTIONAL operation.

Returns a paged listing of test sessions for the current user. Each element in the `data` array is a `test session object` as described in Section 10.16.3. See also Section 10.5.2 for a description of a paged response.

10.16.2. Create a New Test Session

POST /testSessions

Create a new Test Session.

10.16.2.1. Request

`algorithms` is an array of algorithm objects. Each algorithm object has the following available properties:

- **algorithm** — `string`, required

Additional properties for each algorithm are based on the algorithm definition available in each sub-specification.

If not provided `isSample`, and `encryptAtRest` default to `false`.

```
[
    {"acvVersion": "{acvp-version}"},
    {
    "isSample" : true,
    "algorithms": [{
        "algorithm": "TEST_ALGO_1",
        "property1": true,
        "property2": ["operation1", "operation2"]
    }]}
]
```

Figure 44

10.16.2.2. Response

`accessToken` is a [RFC 7519] which **MUST** be supplied as described in Section 10.3 in order to access the Test Session.

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "url": "/acvp/v1/testSessions/2",
        "acvpVersion": "{acvp-version}",
        "createdOn": "2018-05-31T12:03:43Z",
        "expiresOn": "2018-06-30T12:03:43Z",
        "encryptAtRest": false,
```

```
        "vectorSetsUrl": "/acvp/v1/testSessions/2/vectorSets",
        "publishable": false,
        "passed": true,
        "isSample": true,
        "accessToken" : "eyJhbGciOiJIUzI1NiIsInR5cCI6Ik (truncated)"
    }
]
```

Figure 45

10.16.3. Test Session Information

GET /testSessions/{testSessionId}

Returns information about the specific Test Session

10.16.3.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "url": "/acvp/v1/testSessions/2",
        "acvpVersion": "{acvp-version}",
        "createdOn": "2018-05-31T12:03:43Z",
        "expiresOn": "2018-06-30T12:03:43Z",
        "encryptAtRest": false,
        "vectorSetsUrl": "/acvp/v1/testSessions/2/vectorSets",
        "publishable": false,
        "passed": true,
        "isSample": true
    }
]
```

Figure 46

10.16.4. Submit For Validation

PUT /testSessions/{testSessionId}

Certify the Test Session for validation.

Associates all of the testing information with the test session. The test session **MUST** be have both `publishable` and `passed` set to `true`.

10.16.4.1. Request

Available properties:

- **moduleUrl** — `string`

- **module** — a Section 10.11, **MAY** be used instead of `moduleUrl`, but **SHOULD** only be used when the goal is to create a new module resource, otherwise use `moduleUrl` to use an existing module.
- **oeUrl** — `string`
- **oe** — an Section 10.12, **MAY** be used instead of `oeUrl`, but **SHOULD** only be used when the goal is to create a new operating environment resource, otherwise use `oeUrl` to use an existing operating environment.
- **algorithmPrerequisites** — array of algorithm prerequiste objects, optional, for any algorithm that has a prerequisite that was not included in testing, the prerequisite **MUST** be provided by adding an element to this array
  - **algorithm** — `string`, name of the algorithm
  - **mode** — `string`, mode of the algorithm, optional, not all algorithms have a mode
  - **prerequisites** — `string`, array of prerequiste objects
- **algorithm** — `string`, required
- **validationId** — `string`, required

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "moduleUrl": "/acvp/v1/modules/20",
        "oeUrl": "/acvp/v1/oes/60",
        "algorithmPrerequisites": [{
            "algorithm": "TEST_ALGO_1",
            "prerequisites": [
                {
                    "algorithm": "TEST_ALGO_0",
                    "validationId": "123456"
                },
                {
                    "algorithm": "TEST_ALGO_0.1",
                    "validationId": "123456"
                }
            ]
        }]
    }
]
```

Figure 47

10.16.4.2. Response

Reply is a request response as described in Section 10.7. If `status` is `approved` the `approvedUrl` returned will be the identifier of the validation resource which was created or updated as a result of this certification.

10.16.5. Cancel Test Session

DELETE /testSessions/{testSessionId}

Delete a test session.

Marks a test session as being cancelled and may be deleted by the server. Further operations with the test session resource may return 404 HTTP Status.

10.16.6. Request Validation Results

GET /testSessions/{testSessionId}/results

Request Validation Results for a Test Session

10.16.6.1. Response

```
[
  {"acvVersion": "{acvp-version}"},
  {
    "passed": false,
    "results": [
        {
          "vectorSetUrl": "/acvp/v1/testSessions/2/vectorSets/1",
          "status": "incomplete"
        },
        {
           "vectorSetUrl": "/acvp/v1/testSessions/2/vectorSets/2",
           "status": "passed"
        }
    ]
  }
]
```

Figure 48

10.17. Vector Sets

The **REQUIRED** properties for vector set resources are:

- **url** — `string`, identifier for this resource
- **vsId** — `number`
- **algorithm** — `string`

- **mode** — `string`
- **testGroups** — array of test group objects,
  - `{varies}` — based on the values of `algorithm` and `mode` there are zero or more test group properties.
  - **testType** — string defined in algorithm extensions outlining the procedure to complete the corresponding test cases.
  - **tgId** — `number`
  - **tests** — array of test objects,
    - **tcId** — `number`
    - `{varies}` — based on the values of `algorithm` and `mode` there are zero or more test properties.

10.17.1. Vectors Set Listing

GET /testSessions/{testSessionId}/vectorSets

Returns a list of Vector Sets for the specific Test Session.

The property returned is:

- **vectorSetUrls** — array of `string`

10.17.1.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
    {"vectorSetUrls": [
        "/acvp/v1/testSessions/2/vectorSets/1",
        "/acvp/v1/testSessions/2/vectorSets/2"
    ]}
]
```

Figure 49

10.17.2. Vector Set Download

GET /testSessions/{testSessionId}/vectorSets/{vectorSetId}

Vector Set download request.

The server will respond with the vector set associated with the vsId for the client to process. The test group content contained in the response will vary depending on the specific sub-specification of the algorithm and testType being tested.

10.17.2.1. Response

```
[
    {"acvVersion": "{acvp-version}"},
```

```
    {
        "vsId": 1,
        "algorithm": "TEST_ALGO_1",
        "revision": "1.0.0",
        "testGroups": [
            {
                "tgId": 1,
                "testGroupProperty1": 1,
                "testType": "type1",
                "tests": [
                    {
                        "tcId": 1,
                        "testCaseProperty1": 1,
                        "testCaseProperty2": "2"
                    },
                    {
                        "tcId": 2,
                        "testCaseProperty1": 3,
                        "testCaseProperty2": "4"
                    }
                    ... additional tests ...
                ]
            },
            ... additional test groups ...
            {
                "tgId": 3,
                "testGroupProperty1": 2,
                "testType": "type2",
                "tests": [{
                    "tcId": 2139,
                    "testCaseProperty3": 10
                }]
            }
            ... additional test groups ...
        ]
    }
]
```

Figure 50

If the server did not have enough time to generate the vector set for a given test session, the server may reply:

```
[
  { "acvVersion": "{acvp-version}" },
  { "vsId": 1,
    "retry" : 30
```

```
  }
]
```

Figure 51

Where:

- **retry** — represents the number of seconds for the client to wait before retrying the request.

The server may set the `retry` value based on the current server load and expected processing time to generate the vector set.

10.17.3. Cancel Testing of a Vector Set

DELETE /testSessions/{testSessionId}/vectorSets/{vectorSetId}

Cancel testing for a specific Vector Set.

There may be cases where a particular vector set may not be cancelled and the entire Test Session will need to be cancelled instead.

10.17.4. Request Validation Results

GET /testSessions/{testSessionId}/vectorSets/{vectorSetId}/results

Request Validation Results for a Vector Set.

When `showExpected` was set to true from a POST/PUT under Section 10.17.5, additional information is provided back to the client for any failing test cases. The additional information includes an "expected" as well as "provided" object that **MAY** be useful in diagnosing issues within the vector set validation.

10.17.4.1. Response

The client will send this request to learn the validation results for an individual vector set. Properties are:

- **vsId** — `number`
- **disposition** — `string`, the overall result for the vector set with:
  - `fail` — indicates at least one test case has failed.
  - `unreceived` — indicates the server has not received responses from the client for all the test cases.
  - `incomplete` — indicates not all tests have been processed by the server, however none have failed thus far.
  - `expired` — indicates not all the test case responses were received from the client prior to expiry.
  - `passed` — indicates all test cases have been processed by the server and have passed.

- **tests** — array of test result objects
  - **tcId** — `number`
  - **result** — `string`, the result for a test case with:
    - `fail` — indicates the test case has failed.
    - `unreceived` — indicates the server has not received a response from the client for the test case.
    - `incomplete` — indicates the server has not processed the test case.
    - `expired` — indicates the client did not send the test case response to the server prior to expiry.
    - `passed` — indicates the test case passed.
  - **reason** — `string`, provides additional detail in case of a `failed` `result` value.
  - **expected** — `object`, provides the value(s) the server expected for the test case.
  - **provided** — `object`, provides the value(s) the client provided for the test case.

```
[
    {"acvVersion": "{acvp-version}"},
    {"results": {
        "vsId": 1437,
        "disposition": "incomplete",
        "tests": [
            {
                "tcId": 12340,
                "result": "passed",
                "reason": ""
            },
            {
                "tcId": 12341,
                "result": "incomplete",
                "reason": ""
            },
            {
                "tcId": 12342,
                "result": "failed",
                "reason": "Algorithm reason XXX"
            }
        ]
    }}
]
```

Figure 52

10.17.5. Submit Results

POST /testSessions/{testSessionId}/vectorSets/{vectorSetId}/results

Initial Submission of Vector Set Test Results.

10.17.5.1. Request

The client will send this request to submit the test results for an individual vector set. Similar to the vector set download the format will vary depending on the specific sub-specification of the algorithm and testType being tested.

```
[
  {"acvVersion": "{acvp-version}"},
  {
    "vsId": 1437,
    "revision": "1.0.0",
    "showExpected": true,
    "testGroups": [{
      "tgId": 1,
      "tests": [{
          "tcId": 12340,
          "testCaseProperty1": "ABCD",
          "testCaseProperty2": "1234"
        },
        {
          "tcId": 12341,
          "testCaseProperty1": "5678",
          "testCaseProperty2": "FEDC"
        }, ...
      ]
    }, ...
    ]
  }
]
```

Figure 53

The `showExpected` property is optional; when included (and set to true) the ACVP server will include additional information within the validation response file described in Section 10.17.4.

10.17.5.2. Response

No content response. Standard HTTP status codes will indicate success or failure of the submission, but do not indicated the disposition of the tests.

10.17.6. Update Results Submission

PUT /testSessions/{testSessionId}/vectorSets/{vectorSetId}/results

Update Vector Set Test Results Submission.

When one or more test cases fails, the client will need to correct the issue in the crypto module and send the responses again. The resending of responses for failed test cases will occur for an entire vector set. Therefore, even if only a single test case in the vector set failed, the client will need to download, process, and upload responses to the server for the entire vector set (presumably after the problem has been corrected in the implementation). The resending of vector set responses **MUST** occur prior to expiry.

10.17.6.1. Request

The request content is identical to the request content described in Section 10.17.5.

10.17.7. Retrieve Expected Results

GET /testSessions/{testSessionId}/vectorSets/{vectorSetId}/expected

Expected Test Results. Expected test results **SHALL** be generated by the server if the isSample test session resource equals true. See Section 10.16.

10.17.7.1. Response

The response is identical to the request content described in Section 10.17.5.

11. Large Submission

When clients have a response which exceeds the `sizeConstraint` in Section 8, this endpoint **MUST** be used to obtain a URI in order to submit the results. The URI **MAY** be a one-time use URI and clients **SHOULD** only attempt to use this URI once regardless of success or failure in submitting. If resubmission is required clients **MUST** re-POST to the /large endpoint in order to obtain the URI to use.

The available properties for large submissions:

- **submissionSize** — `number`, the maximum size in bytes of the desired submission.
- **vectorSetUrl** — `string`, an identifier of the vector set that will be uploaded as a large submission.
- **url** — `string`, a one-time use url value to use for a large submission.
- **accessToken** — `string`, an optional JWT, that if provided, clients **MUST** use when accessing the `url`. If not provided, clients **MUST** use their current `accessToken` when accessing the `url`.

11.1. Request

POST /large

```
[
  {"acvVersion": "{acvp-version}"},
  {
      "submissionSize": 5000000,
      "vectorSetUrl" : "/acvp/v1/testSessions/1/vectorSets/10"
  }
]
```

Figure 54

11.2. Response

```
[
  {"acvVersion": "{acvp-version}"},
  {
      "url": "https://acvts.authority.org:3954/large/QWERTY123",
      "accessToken" : "<jwt token>"
  }
]
```

Figure 55

12. Vector Set Expiration

Vector sets can expire. For example, in terms of a validation authority use, the vector sets are one-time use only. Old vector sets can never be reused to obtain a new validation certificate for an algorithm implementation or to update an existing certificate. Expiration is a server specific definition which depends on database costs, need for artifacts, etc. If the vector set has expired, the server will reply with an expired response when the client attempts to download the vector set:

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "vsId": 42,
        "status": "expired"
    }
]
```

Figure 56

The ACVP protocol requires server implementations to generate test values and retain the data while the ACVP client processes and returns the results. Some crypto modules implementing the client-side ACVP protocol may not return results immediately. The ACVP protocol design implies the server must retain the test values to verify the client test responses at some time in the future. However, some test vector sets are fairly large, which could place significant storage requirements on ACVP server implementations. To alleviate long term storage requirements, ACVP allows for an expiration timestamp to be set when a test vector set is generated by the server.

The vector set expiration timestamp must be included by the server in the vector set when the client downloads the vector set. The server may change the expiration timestamp of a previously issued vector set to extend its lifetime subject to server policy. The expiration timestamp must be in the ‘expiry’ JSON value, which is included in the JSON encoded vector set. The expiry JSON value will be a string value of the UTC timestamp using form “YYYY-MM-DD HH:MM:SS”. The following figure shows a partial JSON encoded vector set that contains the expiry value.

```
[
    {"acvVersion": "{acvp-version}"},
    {
        "vsId": 1437,
        "expiry": "2018-12-31 23:59:59",
        "algorithm": "TEST_ALGO_1",
        "revision": "1.0.0",
        "testGroups": [
            {
                "tgId": 1,
                "testGroupProperty1": 1,
                "testType": "type1",
```

```
                "tests": [
                    {
                        "tcId": 1,
                        "testCaseProperty1": 1,
                        "testCaseProperty2": "2"
                    },
                    {
                        "tcId": 2,
                        "testCaseProperty1": 3,
                        "testCaseProperty2": "4"
                    },
            .
            .
            .
            <remainder of vector set omitted from figure>
            .
            .
            .
            ]
        }]
    }
]
```

Figure 57

13. Error Codes

Errors will follow HTTP[S] numbering scheme. In addition errors as well as 200 messages may carry JSON encoded information that describes in detail the error and any associated troubleshooting information. Examples of client and server error messages are in Appendix B.

14. Algorithm Test Extensions

ACVP is intended to be an extensible protocol that supports testing of a large number of cryptographic algorithms from several different classes defined by the community. All algorithm identifiers intended for public use **SHALL** be documented by IANA in the ACVP IANA Registry [ACVP IANA Registry].

To add testing for a new algorithm first try to find an algorithm of the same type that is already supported by the protocol.

If it belongs to an already-supported type, check the test specification for the similar algorithm. Typically, similar algorithms share similar testing methodology.

For example, the testing of symmetric block ciphers is comprised of two test types: Algorithm Functional Tests and Monte Carlo Tests—see [ACVP-Symmetric].

Assuming that the existing test types provide sufficient test coverage for the new algorithm, one needs to add the new block cipher algorithm to the symmertic block cipher specification [ACVP-Symmetric], including the JSON schema for the corresponding test data exchanges between the validation server and the client. See in particular Section "Adding new algorithms" in the corresponding algorithm specification.

Next, one needs to update the IANA registry with the new algorithm by adding it to the corresponding namespace and subject to the policies stated in [ACVP IANA Registry].

Once this is completed and the corresponding server test generation and validation for that algorithm are implemented, testing can commence. Clients implementing that algorithm may register it for testing as described in Section "Capabilities Registration" in [ACVP-Symmetric], process the test vectors generated by the validation server and return the results for validation.

If the available test types for an algorithm, existing or new, in a given class do not provide good test coverage of the algorithm, one could develop a new test type and incorporate it into the corresponding test specification for the that algorithm. See for example Section "Adding new algorithms" in [ACVP-Symmetric] for how to add a new test type. Note that this action would require modifications of the corresponding algorithm test specification and would result in a new version of that test specification to be reflected in the IANA registry [ACVP IANA Registry].

15. Custom Specification Objects

15.1. Date

A date type is a time `string` formatted according to the rules of [RFC 3339]; all date/times must use UTC time denoted by 'Z' suffix with no local timezone adjustment. Example is `2018-06-01T20:10:33Z`

15.2. BitString

Bitstrings are used to communicate a string of bits between the ACVP server and IUT.

15.2.1. Endianness

BitStrings **SHALL** be considered in big endian order, unless otherwise specified by the algorithm.

The hex string "FA" (assuming all bits are considered) **SHALL** represent the bits 11111010 (in MSb) or the value 250.

15.2.2. Hex to Bitstring Parsing

"valueLen" will be used as the example, but it can apply to any bit length registration/vector set/ etc parameters.

When a "value" is provided along with a "valueLen", the "valueLen" **MUST** be considered when parsing the hex string represented in "value", **EXCEPT** in empty bitstring cases, which **MUST** be represented as an empty string "". Parsing Hex strings into Bit strings is especially important for algorithms such as the SHA variations that may only include a portion of bits from the provided hex string. When only a portion of bits from a Hex string are considered, bits for use in the bitstring **SHALL** be taken from the most significant bits, meaning the lesser significant bits are the bits that are dropped.

15.2.2.1. Hex string parsing examples

- valueLen: 8, value: "FA", **SHALL** be parsed as the bits 11111010, or the value 250.
- valueLen: 7, value: "FA", **SHALL** be parsed as the bits 1111101, or the value 125.
- valueLen: 5, value: "FA", **SHALL** be parsed as the bits 11111, or the value 31.
- valueLen: 3, value: "FA", **SHALL** be parsed as the bits 111, or the value 7.
- valueLen: 0, value: "", **SHALL** be interpreted as an empty bit string.

15.3. Range

The Range object can be used to convey a range of values. It contains its own set of properties made up of "min", "max", and "increment". "inc" may be used as a shorthand for "increment." If the "increment" property is omitted, a default value of 1 **SHALL** be used.

15.3.1. Range JSON examples

A range object specifying a minimum of 1, a maximum of 8, with an increment of 1. This range object includes the values 1, 2, 3, 4, 5, 6, 7, and 8.

```
{"myRange": {
    "min": 1,
    "max": 8,
    "increment": 1
}}
```

Figure 58

A range object specifying a minimum of 0, a maximum of 8, with an increment of 2. This range object includes the values 0, 2, 4, 6, and 8.

```
{"myRange": {
    "min": 0,
    "max": 8,
    "increment": 2
}}
```

Figure 59

15.4. Domain

The Domain object can be used to specify a set of values similar to Range, albeit with more control. A domain can be made up of an array of objects, where those objects can be literal values, and/or Range objects.

15.4.1. Domain JSON examples

Several sample domain objects that state 0, 8, 16, 32, 96, 128, 192, and 256 are valid values.

```
{"myDomain": [
    {
        "min": 0,
        "max": 16,
        "increment": 8
    },
    32,
    96,
    {
        "min": 128,
        "max": 256,
        "increment": 64
    }
```

```
]}
```

Figure 60

```
{"myDomain": [ 0, 8, 16, 32, 96, 128, 192, 256 ]}
```

Figure 61

15.4.2. Additional Domain Information

Because the Domain is an array of objects consisting of (potentially) both literals and ranges, algorithms that use an array of integers can be used interchangably with the Domain object.

16. Other Considerations

When an ACVP client is attached to a cryptographic module that is in use, access to ACVP **MUST** be controlled so that only an administrator or other authorized user can send and receive ACVP messages. This is because an attacker that has access to ACVP can potentially use it to probe for weaknesses in the cryptographic module.

17. Contributors

Original ACV Protocol created by David McGrew, Bill Hudson and Anthony Grieco of Cisco Systems. Additional contributions made by Sam Farthing, Ellie Daw and Philip Perricone from Cisco Systems and Christopher Celi and Russell Hammett from NIST.

18. JSON Formatting Guidelines

All JSON keywords **SHALL** use lower camelCase format with no underscores or hyphens and use the following characters only a-z, A-Z, 0-9. Keywords SHALL abbreviate common words and phrases wherever possible for brevity.

For example: password length—pwLen plain text length—ptLen

Keywords **SHOULD** be chosen such that they are informative and brief, for example:

```
[
    { "acvVersion": "{acvp-version}" },
    { "results" : { "disposition" : "incomplete" } }
]
```

Figure 62

Metadata assigned to the keyword may use any format which best reflects the information being represented including hyphens, underscores alternating case, numbers, etc. However, brevity should be a major consideration, for example:

```
{
    "algorithms" : [
        { "algorithm" : "ACVP-AES-GCM", "mode" : "modes", "ivGen" :
 "internal", "ivGenMode" : "8.2.1" }
    ]
}
```

Figure 63

All metadata representing strings or big numbers **SHALL** use double quotes at both ends. Big numbers require conversion from strings to whatever format is used by the DUT. Numerical values of integer size or with decimal points may use quotations if those values are generally used as a string, for example the acvVersion would generally be used in displaying information not in any mathematical operations. Something like keyLen or ptLen values would be better used without quotes to avoid having to convert the string to an integer for use in the code.

19. Error Messages

General or registration errors detected by the server **SHALL** result in an HTML error and description of the problem, for example:

HTTP response: 400 "error" : "Incorrectly formatted JSON (51:18): expected field name was not provided: inBit"

Figure 64

Errors detected by the client **SHOULD** trigger an indication of the operation that failed and a detailed error description. This information can be sent to the clients local logging facility to provide traceability of communication issues, for example:

ACV Operation: SHA-512 Error: Unsupported hash algorithm

Figure 65

Appendix A — 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.

G. Klyne, C. Newman (July 2002) *Date and Time on the Internet: Timestamps* (Internet Engineering Task Force), RFC 3339, July 2002. RFC 3339. RFC RFC3339. DOI 10.17487/RFC3339. https://www.rfc-editor.org/info/rfc3339.

T. Berners-Lee, R. Fielding, L. Masinter (January 2005) *Uniform Resource Identifier (URI): Generic Syntax* (Internet Engineering Task Force), STD 66, January 2005. RFC 3986. RFC RFC3986. DOI 10.17487/RFC3986. https://www.rfc-editor.org/info/rfc3986.

T. Dierks, E. Rescorla (August 2008) *The Transport Layer Security (TLS) Protocol Version 1.2* (Internet Engineering Task Force), RFC 5246, August 2008. RFC 5246. RFC RFC5246. DOI 10.17487/RFC5246. https://www.rfc-editor.org/info/rfc5246.

T. Bray(Ed.) (March 2014) *The JavaScript Object Notation (JSON) Data Interchange Format* (Internet Engineering Task Force), RFC 7159, March 2014. RFC 7159. RFC RFC7159. DOI 10.17487/RFC7159. https://www.rfc-editor.org/info/rfc7159.

R. Fielding, J. Reschke (Eds.) (June 2014) *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing* (Internet Engineering Task Force), RFC 7230, June 2014. RFC 7230. RFC RFC7230. DOI 10.17487/RFC7230. https://www.rfc-editor.org/info/rfc7230.

M. Jones, J. Bradley, N. Sakimura (May 2015) *JSON Web Token (JWT)* (Internet Engineering Task Force), RFC 7519, May 2015. RFC 7519. RFC RFC7519. DOI 10.17487/RFC7519. https://www.rfc-editor.org/info/rfc7519.

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.

Vassilev A (November 01, 2018) *ACVP IANA Registry*, November 01, 2018. NIST ACVP IANA Registry.

Vassilev A (February 2018) *Cryptographic Algorithm Validation Program*, February 2018. NIST CAVP.

Celi C, Hammett R (December 10, 2020) *ACVP Symmetric Algorithm JSON Specification* (National Institute of Standards and Technology, Gaithersburg, MD), December 10, 2020.