Automated Cryptographic Validation Protocol
draft-fussell-acvp-spec-00

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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 25, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 4
2. Requirements Language .................................... 4
3. Overview .................................................... 4
   3.1. Audience ............................................. 5
   3.2. Goals ............................................... 5
4. Architecture .............................................. 6
   4.1. Server/Client Architecture(realtime) ................. 6
   4.2. Server/Client Architecture(not realtime) ............. 7
   4.3. Server/Proxy Architecture ........................... 7
   4.4. Terminology ......................................... 8
5. ACV Protocol .............................................. 9
   5.1. HTTP URI Hierarchy .................................. 10
   5.2. HTTP URI Resources .................................. 10
6. Security Considerations .................................. 13
   6.1. Authentication ...................................... 13
7. Encoding ................................................. 14
8. Submission Size Considerations ........................... 14
9. Versioning ............................................... 14
10. Messaging and Workflow ................................ 14
   10.1. Product Registration/Capabilities Exchange .......... 14
   10.2. Test Exchange ...................................... 14
   10.3. JSON Web Token (JWT) ................................. 15
       10.3.1. Authorization flows with JWT .................... 15
       10.3.2. JWT Expiration/Renewal .......................... 16
   10.4. Message Flow ....................................... 17
   10.5. Vendor Resources ................................... 18
       10.5.1. Vendor Listing .................................. 19
       10.5.2. Create a New Vendor ............................. 20
       10.5.3. Vendor Information ............................... 21
       10.5.4. Update an existing Vendor ....................... 22
       10.5.5. Remove a Vendor .................................. 23
   10.6. Modules ............................................ 23
       10.6.1. List Modules .................................... 24
       10.6.2. Register a new Module ........................... 24
       10.6.3. Retrieve information for a Module ............... 25
       10.6.4. Update a Module ................................ 25
       10.6.5. Delete a Module .................................. 26
   10.7. Operational Environments (OEs) ...................... 26
       10.7.1. List Operational Environments ................... 26
       10.7.2. Create a new Operational Environment ............ 27
       10.7.3. Retrieve information for an Operational Environment 28
1. Introduction

The ACV Protocol (ACVP) introduces a method to perform cryptographic assessment and validations at a rate which meets typical industry development cycles. This provides the ability to deploy validated crypto with CVE fixes much faster than previous methods. This document describes how it is structured with respect to the client-server model, the messaging protocol, optional features and flows.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

3. 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
[RFC7230]), 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.

3.1. 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

3.2. 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 [RFC7230], TLS 1.2 [RFC5246] 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.

4. 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 [RFC7159]) 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.

4.1. Server/Client Architecture (realtime)
4.2. Server/Client Architecture (not realtime)

4.3. Server/Proxy Architecture
4.4. 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.
o 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.

o 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.

o OE - Operational Environment - The specific hardware and/or software the client’s cryptographic implementation uses to run.

o 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.

5. ACV Protocol

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

Protocol Layering

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

Figure 4
5.1. HTTP URI Hierarchy

```
+-----------------+---------------+---------------+
| server          | path prefix   | resource      |
+-----------------+---------------+---------------+
| https://acvts.nist.gov/acvp/validation/acvp/login |
+-----------------+---------------+---------------+
| context         | API           |               |
```

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.

5.2. 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>
<thead>
<tr>
<th>Resource</th>
<th>GET (read)</th>
<th>POST (create)</th>
<th>PUT (update)</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>/vendors</td>
<td>Returns a list of vendors (Section 10.5.1)</td>
<td>Register a new vendor (Section 10.5.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/vendors/{vendorId}</td>
<td>Retrieve information for a specific vendor (Section 10.5.3)</td>
<td>Update a vendor (Section 10.5.4)</td>
<td>Delete a vendor (Section 10.5.5)</td>
<td></td>
</tr>
<tr>
<td>/oes</td>
<td>Return a list of OEs (Section 10.5.6)</td>
<td>Create a new OE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/oes/{oeId}</td>
<td>Retrieve Information for a OE (Section 10.7.2)</td>
</tr>
<tr>
<td>/modules</td>
<td>Return a list of Modules (Section 10.6.2)</td>
</tr>
<tr>
<td>/modules/{moduleId}</td>
<td>Retrieve Information for a specific module (Section 10.6.3)</td>
</tr>
<tr>
<td>/dependencies</td>
<td>Returns a list of dependencies (Section 10.8.1)</td>
</tr>
<tr>
<td>/dependencies/properties</td>
<td>Returns a list of properties for dependencies (Section 10.8.3)</td>
</tr>
<tr>
<td>/dependencies/{dependencyId}</td>
<td>Retrieve Information for a specific dependency (Section 10.8.4)</td>
</tr>
<tr>
<td>/algorithms</td>
<td>Return a list of available algorithms (Section 10.9.1)</td>
</tr>
<tr>
<td>/algorithms/{algorithmId}</td>
<td>Retrieve Information about an Algorithm</td>
</tr>
<tr>
<td>Method</td>
<td>Path</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td><code>/testSession s</code></td>
<td>Returns a list of Test Sessions for the current user</td>
</tr>
<tr>
<td><code>/testSession s/{testSession onId}</code></td>
<td>Returns in formation about the specific Test Session for the current user</td>
</tr>
<tr>
<td><code>/testSession s/{testSession onId}/result s</code></td>
<td>Request Validation Results for a Test Session</td>
</tr>
<tr>
<td><code>/testSession s/{testSession onId}/vector Sets</code></td>
<td>Returns a list of Vector Sets for the specific Test Session</td>
</tr>
<tr>
<td><code>/testSession s/{testSession onId}/vector Sets/{vector SetId}</code></td>
<td>Vector Set download request</td>
</tr>
<tr>
<td><code>/testSession s/{testSession onId}/vector Sets/{vector SetId}</code></td>
<td>Vector Set download request for a specific Vector Set Test Results for a specific Test Set Results for a specific Test Set Results for a specific Test Set Results</td>
</tr>
</tbody>
</table>
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

6. 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.

6.1. 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. For the purposes of the message flow examples, no authentication will be used.
7. Encoding

The encoding used for the request/response messaging will be JSON ([RFC7159]). 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.

8. Submission Size Considerations

Standard submission size handled by the server is 4MB. In the case that the submission size is larger than 4MB, an additional port will be provided for the target end point. In order to retrieve this "large submission" URI, the user must calculate the size of the submission in bytes and POST {submissionSize:bytes} to /acvp/acvp/large with their JWT authentication token. This end point will determine if the submission requires the "large submission" URI and if so, will return the URI to the user.

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 5.2 to register and provide cryptographic capabilities. This exchange will consist of several message exchanges and carry the Company name, primary contact (OE information) as well as a detailed list of the supported cryptographic algorithms and their options to be tested during the testSession, see Figure 8.

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 has the option to process the vectorSet.
and return results (realtime) or request more vectorSets for processing at a later time (non-realtime). The client will repeat this process until all of the tests in the session test 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 8.

10.3. JSON Web Token (JWT)

JSON Web Token is described in RFC 7519 [RFC7519] and is used as an authorization mechanism for gaining access to different resources.

An example JWT:

```json
{
  "alg": "none"
}
{
  "iss": "nist.gov",
  "exp": 1426420800,
  "company": "MyCompany",
  "jti": 0987654321,
  "pkey": "cc74f56acdba635079383a03941d68db55c7b3c2f (truncated)"
}

Empty octet string (since alg is none).
```

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.

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 10.10.2

10.3.1. Authorization flows with JWT

JSON Web Token is described in RFC 7519 [RFC7519] 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)"

Workflow authorization flows. All exchanges shown are over HTTP.

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST to /login or similar with appropriate credentials</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive the access token</td>
<td>&lt;- - - - - - - - - -</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6

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 7 below.

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

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST to /login or similar with appropriate credentials and expired JWT access token</td>
<td></td>
<td>session or login JWT</td>
</tr>
<tr>
<td></td>
<td>receive the renewed access token</td>
<td>&lt;- - - - - - - - - -</td>
</tr>
</tbody>
</table>

Figure 7
### 10.4. Message Flow

The minimum message flow between client and server after receiving the JWT: Figure 8

Workflow message flows. All exchanges shown are over HTTP.

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /vendors</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vendor URL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-</td>
<td>- - - - - - - - - -</td>
</tr>
<tr>
<td>POST /modules</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>module URL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-</td>
<td>- - - - - - - - - -</td>
</tr>
<tr>
<td>POST /oes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>oes URL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-</td>
<td>- - - - - - - - - -</td>
</tr>
<tr>
<td>POST dependency</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dependency URL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-</td>
<td>- - - - - - - - - -</td>
</tr>
<tr>
<td>POST testSessions</td>
<td></td>
<td>Submit Registration</td>
</tr>
<tr>
<td></td>
<td>testSessions URL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;-</td>
<td>- - - - - - - - - -</td>
</tr>
<tr>
<td>GET</td>
<td>testSessions/1/vectorSets/1</td>
<td>Retrieve Request</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POST results</td>
<td></td>
<td>Submit</td>
</tr>
</tbody>
</table>
10.5. Vendor Resources

The available properties for vendor resources are:

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

* `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"s

* `contacts* - an array of contact objects,

  * `name* - "string"

  * `emails* - array of "string"s

  * `phoneNumbers* - array of phone objects,

    * `number* - "string"

    * `type* - "string", one of (fax, voice)

* `address* - an address object,

  * `street* - "string"

  * `locality* - "string"

  * `region* - "string"

  * `country* - "string"
10.5.1. Vendor Listing

*GET /vendors*

Returns a listing of vendors.

10.5.1.1. Parameters

Paging Parameters (Section 12)

10.5.1.2. Response
["acvVersion": <acvp-version>],
"vendors": [
{
   "url": "/acvp/v1/vendors/1",
   "name": "Cisco",
   "website": "www.cisco.com",
   "contacts": [{
      "name": "John Doe",
      "email": "johndoe@cisco.com"
   }]
},
{
   "url": "/acvp/v1/vendors/2",
   "name": "Acme, LLC",
   "website": "www.acme.acme",
   "emails": ["inquiry@acme.acme"],
   "contacts": [{
      "name": "Jane Smith",
      "emails": ["jane.smith@acme.acme"],
      "phoneNumbers": [
         {
            "name": "555-555-0001",
            "type": "fax"
         },
         {
            "name": "555-555-0002",
            "type": "voice"
         }
      ],
      "address": {
         "street": "123 Main Street",
         "locality": "Any Town",
         "region": "AnyState",
         "country": "USA",
         "postalCode": "123456"
      }
   }]
}]

10.5.2. Create a New Vendor

*POST /vendors*

Request the creation of a new Vendor.
10.5.2.1. Request

"name" is required and all other properties are OPTIONAL. "url" is not allowed.

```json
[
{
"acvVersion": "<acvp-version>",
"name": "Acme, LLC",
"website": "www.acme.acme",
"emails": [ "inquiry@acme.acme" ],
"contacts": [{
"name": "Jane Smith",
"emails": ["jane.smith@acme.acme"],
"phoneNumbers": [
{
"name": "555-555-0001",
"type": "fax"
},
{
"name": "555-555-0002",
"type": "voice"
}
],
"address": {
"street": "123 Main Street",
"locality": "Any Town",
"region": "AnyState",
"country": "USA",
"postalCode": "123456"
}
}]
}
]
```

10.5.2.2. Response

```json
[
{
"acvVersion": "<acvp-version>",
"url": "/acvp/v1/vendors/2"
}
]
```

10.5.3. Vendor Information

*GET /vendors/{vendorId}*

Retrieve Information for a specific vendor
10.5.3.1. Response

```json
[
  {
    "acvVersion": <acvp-version>,
    "url": "/acvp/v1/vendors/2",
    "name": "Acme, LLC",
    "website": "www.acme.acme",
    "emails": ["inquiry@acme.acme"],
    "contacts": [{
      "name": "Jane Smith",
      "emails": ["jane.smith@acme.acme"],
      "phoneNumbers": [
        {
          "name": "555-555-0001",
          "type": "fax"
        },
        {
          "name": "555-555-0002",
          "type": "voice"
        }
      ],
      "address": {
        "street": "123 Main Street",
        "locality": "Any Town",
        "region": "AnyState",
        "country": "USA",
        "postalCode": "123456"
      }
    }]
  }
]
```

10.5.4. Update an existing Vendor

*PUT /vendors/{vendorId}*

Update a vendor

The "url" property is not updateable.

10.5.4.1. 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"},
  {
    "name": "Acme, LLC",
    "website": "www.acme.acme",
    "emails": [ "inquiry@acme.acme" ],
    "contacts": [{
      "name": "Jane Smith",
      "emails": ["jane.smith@acme.acme"],
      "phoneNumbers": [
        {
          "name": "555-555-0001",
          "type": "fax"
        },
        {
          "name": "555-555-0002",
          "type": "voice"
        }
      ],
      "address": {
        "street": "123 Main Street",
        "locality": "Any Town",
        "region": "AnyState",
        "country": "USA",
        "postalCode": "123456"
      }
    }]
  }
]

10.5.5. Remove a Vendor

*DELETE /vendors/{vendorId}*

Delete a specific vendor.

The server is not required to remove the resource but MUST return an error if the resource will not be removed.

10.6. 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

*vendorUrl* - "string", identifier for a vendor resource (Section 10.5)

*implementationDescription* - "string", a description of the implementation

10.6.1. List Modules

*GET /modules*

Returns a list of available modules.

10.6.1.1. Response

```json
[
  {
    "acvVersion": <acvp-version>},
  {
    "modules": [
      {
        "url": "/acvp/v1/modules/1",
        "name": "ACME ACV Test Module",
        "version": "1.0",
        "type": "Software",
        "vendorUrl": "/acvp/v1/vendors/2",
        "implementationDescription": "ACME module."
      },
      {
        "url": "/acvp/v1/modules/2",
        "name": "ACME ACV Test Module",
        "version": "2.0",
        "type": "Software",
        "vendorUrl": "/acvp/v1/vendors/2",
        "implementationDescription": "ACME module with features"
      }
    ]
  }
]
```

10.6.2. Register a new Module

*POST /modules*

Register a new module.
10.6.2.1. Request

```json
{
    "acvVersion": <acvp-version>,
    "name": "ACME ACV Test Module",
    "version": "3.0",
    "type": "Software",
    "vendorUrl": "/acvp/v1/vendors/2",
    "implementationDescription": "ACME module with more"
}
```  

10.6.2.2. Response

```json
[
    {
        "acvVersion": <acvp-version>,
        "url": "/acvp/v1/modules/5"
    }
]
```  

10.6.3. Retrieve information for a Module

*GET /modules/{moduleId}*

Returns information about a specific module.

10.6.3.1. Response

```json
[
    {
        "acvVersion": <acvp-version>,
        "url": "/acvp/v1/modules/2",
        "name": "ACME ACV Test Module",
        "version": "2.0",
        "type": "Software",
        "vendorUrl": "/acvp/v1/vendors/2",
        "implementationDescription": "ACME module with features."
    }
]
```  

10.6.4. 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.
10.6.4.1. Request

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

10.6.5. Delete a Module

*DELETE /modules/{moduleId}*

Delete a module or mark as no longer in use. (Optional)

10.7. Operational Environments (OEs)

The available properties for operational environment resources are:

*url* - "string", identifier for this resource

*name* - "string"

*dependencyUrls* - an array of "string"s which identify the dependencies (Section 10.8) which comprise this OE.

10.7.1. List Operational Environments

*GET /oes*

Returns a list of available operational environments.

10.7.1.1. Response
10.7.2. Create a new Operational Environment

*POST /oes*

Create a new operational environment.

10.7.2.1. Request


```
[  
  {"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"
    ]
  }
]
```

10.7.2.2. Response


```
[  
  {"acvVersion": <acvp-version>},
  {"url": "/acvp/v1/oes/20"}
]
```
10.7.3. Retrieve information for an Operational Environment

*GET /oes/{oeId}*

Returns information about a specific operational environment.

10.7.3.1. Response

```json
[
  {
    "acvVersion": <acvp-version>,
    {
      "url": "/acvp/v1/oes/1",
      "name": "Windows 10 on Intel Xeon 5100 Series Processor",
      "dependencyUrls": [
        "/acvp/v1/dependencies/1",
        "/acvp/v1/dependencies/2"
      ]
    }
  }
]
```

10.7.4. 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.

10.7.4.1. Request

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

10.7.5. Delete an Operational Environment

*DELETE /oes/{oeId}*

Delete an operational environment or mark as no longer in use. 
(Optional)
10.8. Dependencies

The available properties for dependency resources are:

*url* - "string", identifier for this resource

*type* - "string"

"{varies}" depending on the value of "type" as defined by the response of Section 10.8.3

10.8.1. List Dependencies

*GET /dependencies*

Returns a list of available dependencies.

10.8.1.1. Response

```
[ {
   "acvVersion": <acvp-version>},
   "dependencies": [ {
      "type": "software",
      "name": "Linux 3.1",
      "cpe": "cpe-2.3:o:ubuntu:linux:3.1"
   },
   {
      "type": "processor",
      "manufacturer": "Intel",
      "family": "ARK",
      "name": "Xeon",
      "series": "5100",
      "features": ["rdrand"]
   }
   ]
]
```

10.8.2. Register a new Dependency

*POST /dependencies*

Register a new dependency.
10.8.2.1. Request

```json
[{
  "acvVersion": <acvp-version>,
  {
    "type": "software",
    "name": "Linux 3.1",
    "cpe": "cpe-2.3:o:ubuntu:linux:3.1"
  }
}
]
```

10.8.2.2. Response

```json
[{
  "acvVersion": <acvp-version>,
  "url": "/acvp/v1/dependencies/7"
}
]
```

10.8.3. List Dependency Properties

*GET /dependencies/properties*

Returns the 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"s where each element corresponds to a dependency type value that this property may be used with.

*description* - "string"

10.8.3.1. Response

```json
[
  {
    "acvVersion": <acvp-version>,
    "properties": [
      {
        "name": "name",
        "dataType": "string",
        "validTypes": [
          "software",
          "processor"
        ]
      }
    ]
  }
]
```
"name": "swid",
"dataType": "string",
"validTypes": ["software"],
},

{"name": "cpe",
"dataType": "string",
"validTypes": ["software", "processor"],
},

{"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."
}]}
10.8.4. Retrieve information for a Dependency

*GET /dependencies/{dependencyId}*

Returns information about a specific dependency.

10.8.4.1. Response

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

10.8.5. 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.

10.8.5.1. Request

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

10.8.6. Delete a Dependency

*DELETE /dependencies/{dependencyId}*

Delete a dependency or mark as no longer in use. (Optional)

10.9. Algorithms

The Algorithm resources are informational only.
10.9.1. Algorithms Listing

*GET /algorithms*

Returns a list of available algorithms on the server.

10.9.1.1. Response

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

10.9.2. Algorithm Information

*GET /algorithms/{algorithmId}*

Retrieve Information for about a specific algorithm.

10.9.2.1. Response

Response may vary from server depending on internal representation.

10.10. Test Sessions

The available properties for test session resources are:

*url* - "string", identifier for this resource
*acvpVersion* - "string", version of ACV protocol used to create the test session.

*createdOn* - date (Section 15.1)

*expiresOn* - date (Section 15.1)

*encryptAtRest* - "boolean"

*vectorSetUrls* - "string"

*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 /testSessions/{testSessionId}/vectorSets/{vectorSetId}/expected (Section 10.11.7) will return expected result values. As well, Test Vector Sets MAY contain fewer Test Cases for quicker generation and verification.

*production* - "boolean", indicates that the test session excludes tests that require intermediate values; this may cause the test session to have a "publishable" value of "false".

10.10.1. Test Session Listing (Current User)

*GET /testSessions*

Returns a list of Test Sessions for the current user.

This is an OPTIONAL operation.

10.10.1.1. Response
["acvVersion": <acvp-version>],
"testSessions": [ {
  "url": "/acvp/v1/testSessions/2",
  "acvVersion": <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,
  "production": false,
  "isSample": true
}]
}

10.10.2. Create a New Test Session

*POST /testSessions*

Create a new Test Session.

10.10.2.1. 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", "production" and "encryptAtRest" default to "false".

[
  {
    "acvVersion": <acvp-version>},
  {
    "isSample": true,
    "algorithms": [{
      "algorithm": "TEST_ALGO_1",
      "property1": true,
      "property2": ["operation1", "operation2"]
    }]
  }
]
10.10.2.2. Response

"accessToken" is a JWT Token [RFC7519] 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>",
    "createdAt": "2018-05-31T12:03:43Z",
    "expiresAt": "2018-06-30T12:03:43Z",
    "encryptAtRest": false,
    "vectorSetsUrl": "/acvp/v1/testSessions/2/vectorSets",
    "publishable": false,
    "passed": true,
    "production": false,
    "isSample": true,
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6Iklc (truncated)"
}
```

10.10.3. Test Session Information

*GET /testSessions/{testSessionId}*

Returns information about the specific Test Session

10.10.3.1. Response

```
{
    "acvVersion": "<acvp-version>",
    "url": "/acvp/v1/testSessions/2",
    "acvpVersion": "<acvp-version>",
    "createdAt": "2018-05-31T12:03:43Z",
    "expiresAt": "2018-06-30T12:03:43Z",
    "encryptAtRest": false,
    "vectorSetsUrl": "/acvp/v1/testSessions/2/vectorSets",
    "publishable": false,
    "passed": true,
    "production": false,
    "isSample": true
}
```
10.10.4. 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".

10.10.4.1. Request

Available properties:

*moduleUrl* - "string"

*oeUrl* - "string"

*signature* - a signature object,

  *algorithm* - "string"

  *certificate* - "string"

  *digitalSignature* - "string"

*algorithmPrerequisites* - array of algorithm prerequisite 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 prerequisite 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"
      }
    ]
  }],
  "signature": {
    "algorithm": "SHA256RSA",
    "certificate": "{base64encodedcertificate}"
    "digitalSignature": "{base64encodedsignature}"
  }
}

10.10.5. 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.

10.10.6. Request Validation Results

*GET /testSessions/{testSessionId}/results*

Request Validation Results for a Test Session

10.10.6.1. Response
10.11. 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.

10.11.1. 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"s

10.11.1. Response

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

10.11.2. 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.

10.11.2.1. Response
If the server did not have enough time to generate the vector set for a given test session, the server may reply:
### 10.11.3. 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.

### 10.11.4. Request Validation Results

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

Request Validation Results for a Vector Set.

#### 10.11.4.1. 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* - "number", provides additional detail in case of a "failed" "result" value.
10.11.5. Submit Results

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

Initial Submission of Vector Set Test Results.

10.11.5.1. 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.
10.11.5.2. Response

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

10.11.6. 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.
10.11.6.1. Request

The request content is identical to the request content described in Section 10.11.5.

10.11.7. Retrieve Expected Results

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

Expected Test Results.

10.11.7.1. Response

The response is identical to the request content described in Section 10.11.5.

11. 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": <vs-id>,
  "status": "expired"
]
```

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.

```json
[
  {"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"
          }
        ]
      }
    ]
  }
]
```
12.  Paging Parameters

Some resources require paging in order to avoid returning large amounts of data. To facilitate paging the following Query parameters SHOULD be allowed on resources where paging is necessary.

"limit" - the number of entries to return

"offset" - the offset into the list of entries

The response will indicate the offset and the total count using "offset" and "total" properties.

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].

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 [sub-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 symmetric block cipher specification [sub-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].

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 [sub-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 [sub-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].

15. Custom Specification Objects

15.1. Date

A date type is a time "string" formatted according to the rules of RFC 3339 [RFC3339]; 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".

15.3.1. Range JSON examples

A range object specifying a minimum of 0, a maximum of 8, with an increment of 1. This range object includes the values 1, 2, 3, 4, 5, 6, 7, and 8.

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

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
  }
]
```

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

The IANA considerations for this memo are provided by [acvp-iana].

17. 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.

18. 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.

19. References

19.1. Normative References


19.2. Informative References


Appendix A. 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:

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

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:

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

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.

Appendix B. 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"

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 client's local logging facility to provide traceability of communication issues, for example:

ACV Operation: SHA-512
Error: Unsupported hash algorithm

Authors' Addresses

Barry Fussell (editor)
Cisco Systems
170 West Tasman Dr.
San Jose, CA  95134
USA

Email: bfussell@cisco.com
Apostol Vassilev (editor)
National Institute of Standards and Technology
100 Bureau Dr.
Gaithersburg, MD  20899
USA

Email: apostol.vassilev@nist.gov

Harold Booth
National Institute of Standards and Technology
100 Bureau Dr.
Gaithersburg, MD  20899
USA

Email: harold.booth@nist.gov