ACVP Secure Hash Algorithm (SHA) JSON Specification

Christopher Celi
Information Technology Laboratory
Computer Security Division

November 01, 2018

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

Abstract

This document defines the JSON schema for testing Secure Hash Algorithm (SHA) implementations with the ACVP specification.

Keywords

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

ACVP; cryptography

Foreword

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Audience

This document is intended for the users and developers of ACVP.

Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 of [RFC 2119] and [RFC 8174] when, and only when, they appear in all capitals, as shown here.

Acknowledgements

This document is produced by the Security Testing, Validation and Measurement group under the Automated Cryptographic Validation Testing (ACVT) program.

Executive Summary

The Automated Crypto Validation Protocol (ACVP) defines a mechanism to automatically verify the cryptographic implementation of a software or hardware crypto module. The ACVP specification defines how a crypto module communicates with an ACVP server, including crypto

capabilities negotiation, session management, authentication, vector processing and more. The ACVP specification does not define algorithm specific JSON constructs for performing the crypto validation. A series of ACVP sub-specifications define the constructs for testing individual crypto algorithms. Each sub-specification addresses a specific class of crypto algorithms. This sub-specification defines the JSON constructs for testing Secure Hash Algorithm (SHA) implementations using ACVP.

Disclaimer

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

Additional Information

For additional information on NIST's Cybersecurity programs, projects and publications, visit the Computer Security Resource Center. Information on other efforts at NIST and in the Information Technology Laboratory (ITL) is also available.

Feedback

Feedback on this publication is welcome, and can be sent to: code-signing@nist.gov.

1. Introduction

The Automated Crypto Validation Protocol (ACVP) defines a mechanism to automatically verify the cryptographic implementation of a software or hardware crypto module. The ACVP specification defines how a crypto module communicates with an ACVP server, including crypto capabilities negotiation, session management, authentication, vector processing and more. The ACVP specification does not define algorithm specific JSON constructs for performing the crypto validation. A series of ACVP sub-specifications define the constructs for testing individual crypto algorithms. Each sub-specification addresses a specific class of crypto algorithms. This sub-specification defines the JSON constructs for testing Secure Hash Algorithm (SHA) implementations using ACVP.

2. Supported Hash Algorithms

The following hash algorithms MAY be advertised by the ACVP compliant cryptographic module:

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

3. Test Types and Test Coverage

This section describes the design of the tests used to validate implementations of SHA-1 and SHA-2.

3.1. Test Types

There are two types of tests for SHA-1 and SHA-2: functional tests and Monte Carlo tests. Each has a specific value to be used in the testType field. The testType field definitions are:

- "AFT" — Algorithm Functional Test. These tests can be processed by the client using a normal 'hash' operation. AFTs cause the implementation under test to exercise normal operations on a single block, multiple blocks, or partial blocks. In all cases, random data is used. The functional tests are designed to verify that the logical components of the hash function (block chunking, block padding etc.) are operating correctly.
- "MCT" — Monte Carlo Test. These tests exercise the implementation under test under stenuous circumstances. The implementation under test must process the test vectors according to the correct algorithm and mode in this document. MCTs can help detect potential memory leaks over time, and problems in allocation of resources, addressing variables, error handling, and generally improper behavior in response to random inputs. Each MCT processes 100 pseudorandom tests. Each algorithm and mode SHOULD have at least one MCT group. See Section 3.2 for implementation details.
- "LDT" — Large Data Test. This test performs the hash function on a message that is multiple gigabytes in length. This pushes the bounds of 32-bit data types to ensure an implementation can handle all types of data. See [LDT] for more information motivating the LDT. As a multiple gigabyte message cannot be communicated naturally via ACVP, a specific structure is outlined in Section 3.3.

3.2. Monte Carlo tests for SHA-1 and SHA-2

The MCTs start with an initial condition (SEED which is a single message) and perform a series of chained computations. The algorithm is shown below.

```
For j = 0 to 99
    A = B = C = SEED
    For i = 0 to 999
        MSG = A || B || C
        MD = SHA(MSG)
        A = B
        B = C
        C = MD
    Output MD
```

```
    SEED = MD
```

Figure 1 — SHA-1 and SHA-2 Monte Carlo Test

3.3. Large Data tests for SHA-1 and SHA-2

The large data tests are intended to test the ability of a module to hash multiple gigabytes of data at once. This much information cannot be communicated via the JSON files as a normal message property. Instead a new type is defined as a large data type. It is an object that contains a small content hex string, a content length in bits, a full length in bits and an expansion technique string. The following is an example of this structure.

```
"largeMsg": {
    "content": "DE26",
    "contentLength": 16,
    "fullLength": 42949672960,
    "expansionTechnique": "repeating"
}
```

Figure 2

The ‘contentLength’ property describes the number of bits in the ‘content’ property. The ‘content’ property is the hex string that can be expanded to the full large message. The ‘expansionTechnique’ describes the process used to obtain the full large message. The ‘fullLength’ is the final length of the full large message.

There may be multiple ‘expansionTechnique’ types defined. Here are the types defined for SHA-1 and SHA-2 testing.

- “repeating” — Append the number of content bits specified in ‘contentLength’ to itself as many times as needed until a hex string of exactly ‘fullLength’ bits is acquired. In the example shown, the final large message would have the form “DE26DE26DE26…DE26”.

There are multiple ways hash functions can be implemented in an IUT. The most common are via a single Hash() call on the message or via a series of Init(), any number of Update(), Final() calls. As noted in [LDT], the difference between these hashing techniques can have consequences in the cryptographic module. If both interfaces are offered and accessible for testing, the IUT **MUST** only utilize a single Update() call for the large message.

3.4. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [FIPS 180-4].

3.4.1. SHA Requirements Covered

Section 1 in [FIPS 180-4] outlines the maximum message sizes for each hash function. Due to the large size (either 2^64 or 2^128 bits) of these maximums, they are tested by special request in this specification. These tests are performed by the optional LDTs.

Sections 3 and 4 in [FIPS 180-4] outline the core functions used within the hash algorithms. Normal AFTs test these operations.

Section 5 outlines the hash function preprocessing. It is worth noting that not all test cases will cover the message padding process, but through the entire vector set, this operation will be fully tested.

3.4.2. SHA Requirements Not Covered

Section 7 outlines digest truncation for applications where a shortened digest is needed. These operations are not tested via this specification.

4. Capabilities Registration

ACVP requires crypto modules to register their capabilities. This allows the crypto module to advertise support for specific algorithms, notifying the ACVP server which algorithms need test vectors generated for the validation process. This section describes the constructs for advertising support of SHA algorithms to the ACVP server.

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

4.1. Prerequisites

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

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

Table 1 — Prerequisite Properties

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

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

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

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

```
]
```

Figure 3

4.2. HASH Algorithm Capabilities Registration

This section describes the constructs for advertising support of hash algorithms to the ACVP server.

Table 2 — Hash Algorithm Capabilities JSON Values

JSON Value	Description	JSON type
algorithm	The hash algorithm and mode to be validated.	string
revision	The algorithm testing revision to use.	string
messageLength	The message lengths in bits supported by the IUT. Minimum allowed is 0, maximum allowed is 65535.	domain
performLargeDataTest	Determines if the server should include the large data test group defined in Section 3.3. This property is **OPTIONAL**, and shall include the lengths in GiB being tested. Valid options are {1, 2, 4, 8}.	integer array

The value of the algorithm property **MUST** be one of the elements from the list in Section 2.

NOTE – The lengths provided in the 'performLargeDataTest' property are in gibibytes. 1GiB is equivalent to 2^30 bytes.

The following is a example JSON object advertising support for SHA-256.

```
{
    "algorithm": "SHA2-256",
    "revision": "1.0",
    "messageLength": [{"min": 0, "max": 65535, "increment": 1}],
    "performLargeDataTest": [1, 2]
}
```

Figure 4

5. Test Vectors

The ACVP server provides test vectors to the ACVP client, which are then processed and returned to the ACVP server for validation. A typical ACVP validation test session would require multiple test vector sets to be downloaded and processed by the ACVP client. Each test vector set represents an individual algorithm defined during the capability exchange. This section describes the JSON schema for a test vector set used with Secure Hash Algorithm (SHA) algorithms.

The test vector set JSON schema is a multi-level hierarchy that contains meta data for the entire vector set as well as individual test vectors to be processed by the ACVP client. The following table describes the JSON elements at the top level of the hierarchy.

Table 3 — Top Level Test Vector JSON Elements

JSON Values	Description	JSON Type
acvVersion	Protocol version identifier	string
vsId	Unique numeric vector set identifier	integer
algorithm	Algorithm defined in the capability exchange	string
mode	Mode defined in the capability exchange	string
revision	Protocol test revision selected	string
testGroups	Array of test groups containing test data, see Section 5.1	array

An example of this would look like this

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

Figure 5

5.1. Test Groups

Test vector sets **MUST** contain one or many test groups, each sharing similar properties. For instance, all test vectors that use the same testType would be grouped together. The testGroups element at the top level of the test vector JSON object **SHALL** be the array of test groups. The Test Group JSON object **MUST** contain meta-data that applies to all test cases within the group. The following table describes the JSON elements that **MAY** appear from the server in the Test Group JSON object:

Table 4 — Test Group JSON Object

JSON Value	Description	JSON type
tgId	Numeric identifier for the test group, unique across the entire vector set	integer
testType	Test category type (AFT, MCT or LDT). See Section 3 for more information	string
tests	Array of individual test case JSON objects, which are defined in Section 5.2	array of testCase objects

All properties described in the previous table MUST appear in the prompt file from the server for every testGroup object.

5.2. Test Case

Each test group **SHALL** contain an array of one or more test cases. Each test case is a JSON object that represents a single case to be processed by the ACVP client. The following table describes the JSON elements for each test case.

Table 5 — Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
len	Length of the message or MCT seed	integer
msg	Value of the message or MCT seed in big-endian hex	integer
largeMsg	Object describing the message for an LDT group	large data object, see Section 3.3 for more information

All properties described in the previous table **MUST** appear in the prompt file from the server for every testCase object.

The following is an example JSON object for secure hash test vectors sent from the ACVP server to the crypto module. Note the single bit message is represented as “80”. This complies with SHA1 and SHA2 being big-endian by nature. All hex strings associated with SHA1 and SHA2 **SHALL** be big-endian.

```
[
    { "acvVersion": <acvp-version> },
    {
        "vsId": 1564,
        "algorithm": "SHA2-512/224",
        "revision": "1.0",
```

```
    "testGroups": [
        {
            "tgId": 1,
            "testType": "AFT",
            "tests": [
                {
                    "tcId": 0,
                    "len": 0,
                    "msg": "00"
                },
                {
                    "tcId": 1,
                    "len": 1,
                    "msg": "80"
                }
            ]
        },
        {
            "tgId": 2,
            "testType": "MCT",
            "tests": [
                {
                    "tcId": 2175,
                    "len": 20,
                    "msg": "331b04d56f6e3ed5af349bf1fd9f9591b6ec886e",
                }
            ]
        },
        {
            "tgId": 3,
            "testType": "LDT",
            "tests": [
                {
                    "tcId": 1029,
                    "largeMsg": {
                        "content": "DE26",
                        "contentLength": 16,
                        "fullLength": 42949672960,
                        "expansionTechnique": "repeating"
                    }
                }
            ]
        }
    ]
}
```

```
]
```

Figure 6

5.3. Test Vector Responses

After the ACVP client downloads and processes a vector set, it **SHALL** send the response vectors back to the ACVP server within the alloted timeframe. The following table describes the JSON object that represents a vector set response.

Table 6 — Vector Set Response JSON Object

JSON Value	Description	JSON type
acvVersion	Protocol version identifier	string
vsId	Unique numeric identifier for the vector set	integer
testGroups	Array of JSON objects that represent each test vector result, which uses the same JSON schema as defined in Section 5.2	array of testGroup objects

The testGroup Response section is used to organize the ACVP client response in a similar manner to how it receives vectors. Several algorithms **SHALL** require the client to send back group level properties in its response. This structure helps accommodate that.

Table 7 — Vector Set Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
tests	The tests associated to the group specified in tgId	array of testCase objects

Each test case is a JSON object that represents a single test object to be processed by the ACVP client.

The following table describes the JSON elements for each test case object.

Table 8 — Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set.	integer
md	The IUT's message digest response to an AFT or LDT test	string (hex)
resultsArray	Array of JSON objects that represent each iteration of	array of objects containing the md

JSON Value	Description	JSON type
	an MCT. Each iteration will output the md	

Note: The tcId **MUST** be included in every test case object sent between the client and the server.

The following is a example JSON object for secure hash test results sent from the crypto module to the ACVP server. The group identified by tgId 1 is a group of AFTs. The group identified by tgId 2 is a group of MCTs. The group identified by tgId 3 is a group of LDTs.

```
{
    "vsId": 0,
    "algorithm": "SHA2-224",
    "revision": "1.0",
    "testGroups": [
        {
            "tgId": 1,
            "tests": [
                {
                    "tcId": 1,
                    "md":
"D14A028C2A3A2BC9476102BB288234C415A2B01F828EA62AC5B3E42F"
                },
                {
                    "tcId": 2,
                    "md":
"D14A028C2A3A2BC9476102BB288234C415A2B01F828EA62AC5B3E42F"
                }
            ]
        },
        {
            "tgId": 2,
            "tests": [
                {
                    "tcId": 1028,
                    "resultsArray": [
                        {
                            "md":
"E82B1FA3D510C2E423D03CEDF01F66C995CDD573EB63D5A33FDAE640"
                        },
                        {
                            "md":
"00179AE4CE57DCBD156B981A414370B5089FA8E1098A05443DF3CD62"
                        },
                        {
```

```
                            "md":
 "8F6C7F546940352613E8389D4F4B87473A57CACD7E289A27E4F51965"
                        }
                    ]
                }
            ]
        },
        {
            "tgId": 3,
            "tests": [
                {
                    "tcId": 1029,
                    "md":
 "E4F8B44B32F5A25D1F4784601BF095CF5F7C6F4E9EAA1005AD19F09A"
                }
            ]
        }
    ]
}
```


Figure 7

6. Security Considerations

There are no additional security considerations outside of those outlined in the ACVP document.

Appendix A — Terminology

For the purposes of this document, the following terms and definitions apply.

A.1.

Prompt
JSON sent from the server to the client describing the tests the client performs
Registration
The initial request from the client to the server describing the capabilities of one or several algorithm, mode and revision combinations
Response
JSON sent from the client to the server in response to the prompt
Test Case
An individual unit of work within a prompt or response
Test Group
A collection of test cases that share similar properties within a prompt or response
Test Vector Set
A collection of test groups under a specific algorithm, mode, and revision
Validation
JSON sent from the server to the client that specifies the correctness of the response

Appendix B — Abbreviations and Acronyms

ACVP	Automated Crypto Validation Protocol
JSON	Javascript Object Notation

Appendix C — Revision History

Table C-1

Version	Release Date	Updates
1	2018-11-01	Initial Release

Appendix D — References

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

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

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

National Institute of Standards and Technology (August 2015) *Secure Hash Standard (SHS)* (Gaithersburg, MD), August 2015. FIPS 180-4. https://doi.org/10.6028/NIST.FIPS.180-4.

Fussell B, Vassilev A, Booth H, Celi C, Hammett R (July 01, 2019) *Automatic Cryptographic Validation Protocol* (National Institute of Standards and Technology, Gaithersburg, MD), July 01, 2019.

Bassham III L (2014) *The Secure Hash Algorithm Validation System (SHAVS)* (National Institute of Standards and Technology, Gaithersburg, MD), 2014.

Extending NIST's CAVP Testing of Cryptographic Hash Function Implementations. LDT.