ACVP SafePrimes JSON Specification

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

June 05, 2019

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

Abstract

This document defines the JSON schema for testing SafePrimes 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 SafePrimes 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 SafePrimes implementations using ACVP.

2. Supported Safe Prime Functions

The following Safe Prime operations **MAY** be advertised by the ACVP compliant cryptographic module listed as “algorithm” / “mode” / “revision”:

- safePrimes / keyGen / 1.0
- safePrimes / keyVer / 1.0

3. Test Types and Test Coverage

The ACVP server performs a set of tests on the specified SafePrimes algorithm in order to assess the correctness and robustness of the implementation. A typical ACVP validation session **SHALL** require multiple tests to be performed for every supported permutation of SafePrimes capabilities. This section describes the design of the tests used to validate implementations of the SafePrimes algorithms.

3.1. Test Types

- SafePrimes / keyGen / 1.0 "AFT" — Algorithm Functional Test. The IUT is **REQUIRED** for each test case provided, to generate a key pair based on a SafePrime group. The IUT generated key pairs are sent to the ACVP server and are checked for validity against the group.
- SafePrimes / keyVer / 1.0 "AFT" — Algorithm Functional Test. The ACVP server generates a series of key-pairs to communicate to the IUT per safe prime group. The IUT is **REQUIRED** to validate the key-pair satisfies `0 < x < q` and `y = g^x mod p`.

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [SP 800-56A Rev. 3], [RFC 3526], and [RFC 7919].

3.2.1. SafePrime Requirements Covered

- SP800-56Ar3 — Section 5.6.1.1.1 Using the Approved Safe-Prime Groups. This ACVP server specification makes testing available/validatable for use in Safe-Prime Groups key generation for KAS-FFC as specified in [SP 800-56A Rev. 3].
- RFC3526 — All safe prime groups defined in this document are made available for testing via the ACVP server.
- RFC7919 — All safe prime groups defined in this document are made available for testing via the ACVP server.

3.2.2. SafePrime Requirements Not Covered

N/A

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 SafePrimes algorithms to the ACVP server.

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

4.1. Prerequisites

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

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

Table 1 — Prerequisite Properties

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

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

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

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

```
]
```

Figure 1

4.2. SafePrime Prerequisites

Each safePrimes / * / * implementation relies on other cryptographic primitives. For example, safePrimes / * / * uses an underlying SHA algorithm. Each of these underlying algorithm primitives must be validated, either separately or as part of the same submission. ACVP provides a mechanism for specifying the required prerequisites:

Table 2 — Required SafePrime Prerequisite Algorithms JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	a prerequisite algorithm	string	DRBG, SHA, SHA_OPT2
valValue	algorithm validation number	string	actual number or “same”
prereqAlgVal	prerequistie algorithm validation	object with algorithm and valValue properties	see above
prereqVals	prerequistie algorithm validations	array of prereqAlgVal objects	see above

4.3. SafePrime Capabilities

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

Table 3 — SafePrime Capabilities JSON Properties

JSON Value	Description	JSON Type	Valid Values
algorithm	The algorithm under test	string	“safePrimes”
mode	The SafePrimes mode to be validated	string	“keyGen” or “keyVer”
revision	The algorithm testing revision to use	string	“1.0”
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See Table 2
safePrimeGroups	Safe prime groups to test with	array of string	Any non-empty subset of {”MODP-2048”, “MODP-3072”, “MODP-4096”, “MODP-6144”, “MODP-8192”, “ffdhe2048”, “ffdhe3072”,

JSON Value	Description	JSON Type	Valid Values
			“ffdhe4096”, “ffdhe6144”, “ffdhe8192”}

4.4. Safe Prime Example Registrations

A safePrime / keyGen / 1.0 registration looks like the following

```
{
 "algorithm": "safePrimes",
 "mode": "keyGen",
 "revision": "1.0",
 "prereqVals": [{
  "algorithm": "DRBG",
  "valValue": "123456"
 }],
 "safePrimeGroups": [
  "ffdhe2048",
  "MODP-2048"
 ]
}
```

Figure 2

A safePrime / keyVer / 1.0 registration looks like the following

```
{
    "algorithm": "safePrimes",
    "mode": "keyVer",
    "revision": "1.0",
    "prereqVals": [{
        "algorithm": "DRBG",
        "valValue": "123456"
    }],
    "safePrimeGroups": [
        "ffdhe2048",
        "MODP-2048"
    ]
}
```

Figure 3

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 SafePrimes 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 4 — Top Level Test Vector JSON Elements

JSON Values	Description	JSON Type
acvVersion	Protocol version identifier	string
vsId	Unique numeric vector set identifier	integer
algorithm	Algorithm defined in the capability exchange	string
mode	Mode defined in the capability exchange	string
revision	Protocol test revision selected	string
testGroups	Array of test groups containing test data, see Section 6	array

An example of this would look like this

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

Figure 4

6. Test Vectors

The ACVP server provides test vectors to the ACVP client, which are then processed and returned to the ACVP server for validation. A typical ACVP validation session would require multiple test vector sets to be downloaded and processed by the ACVP client. Each test vector set represents an individual crypto algorithm, such as SafePrimes / keyVer / 1.0, etc. This section describes the JSON schema for a test vector set used with Safe Primes crypto 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 5 — Safe Primes Vector Set JSON Object

JSON Value	Description	JSON type
acvVersion	Protocol version identifier	string
vsId	Unique numeric identifier for the vector set	integer
algorithm	The algorithm used for the test vectors	string
mode	The mode used for the test vectors	string
revision	The algorithm testing revision to use	string
testGroups	Array of test group JSON objects, which are defined in Section 6.1, or Section 6.2 depending on the algorithm	array

6.1. SafePrime keyGen Test Vectors

6.1.1. SafePrime keyGen Test Groups JSON Schema

The testGroups element at the top level in the test vector JSON object is an array of test groups. Test vectors are grouped into similar test cases to reduce the amount of data transmitted in the vector set. For instance, all test vectors that use the same key size would be grouped together. The Test Group JSON object contains meta data that applies to all test vectors within the group. The following table describes the secure hash JSON elements of the Test Group JSON object.

The test group for safePrime / keyGen / 1.0 is as follows:

Table 6 — Safe Prime keyGen Test Group JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
testType	The test operation performed	string
safePrimeGroup	The safe prime group that the IUT should use for generating keys	string

JSON Value	Description	JSON type
tests	Array of individual test vector JSON objects, which are defined in Section 6.1.2	array

6.1.2. SafePrime keyGen Test Case JSON Schema

Each test group contains an array of one or more test cases. Each test case is a JSON object that represents a single test vector to be processed by the ACVP client. The following table describes the JSON elements for each safePrime / keyGen / 1.0 test vector.

Table 7 — SafePrime keyGen Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
deferred	States that the values are generated by the client	boolean
NOTE – The client is responsible for generating a key to be verified by the server.		

The following is an example JSON object sent from the server to the client for safePrimes / keyGen / 1.0.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 0,
        "algorithm": "safePrimes",
        "mode": "keyGen",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "safePrimeGroup": "ffdhe2048",
                "testType": "AFT",
                "tests": [
                    {
                        "tcId": 1
                    },
                    {
                        "tcId": 2
                    },
                    {
                        "tcId": 3
```

```
                    }
                ]
            },
            {
                "tgId": 2,
                "testType": "AFT",
                "safePrimeGroup": "MODP-2048",
                "tests": [
                    {
                        "tcId": 4
                    },
                    {
                        "tcId": 5
                    },
                    {
                        "tcId": 6
                    }
                ]
            }
        ]
    }
]
```

Figure 5

6.2. SafePrime keyVer Test Vectors

6.2.1. SafePrime keyVer Test Groups JSON Schema

The testGroups element at the top level in the test vector JSON object is an array of test groups. Test vectors are grouped into similar test cases to reduce the amount of data transmitted in the vector set. For instance, all test vectors that use the same key size would be grouped together. The Test Group JSON object contains meta data that applies to all test vectors within the group. The following table describes the secure hash JSON elements of the Test Group JSON object.

The test group for safePrime / keyVer / 1.0 is as follows:

Table 8 — Safe Prime keyVer Test Group JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
testType	The test operation performed	string
safePrimeGroup	The safe prime group that the IUT should use for validating keys	string

JSON Value	Description	JSON type
tests	Array of individual test vector JSON objects, which are defined in Section 6.2.2	array

6.2.2. SafePrime keyVer Test Case JSON Schema

Each test group contains an array of one or more test cases. Each test case is a JSON object that represents a single test vector to be processed by the ACVP client. The following table describes the JSON elements for each safePrime / keyVer / 1.0 test vector.

Table 9 — SafePrime keyVer Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
x	The private key component X	hex
y	The public key component Y	hex

The following is an example JSON object sent from the server to the client for safePrimes / keyVer / 1.0.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 0,
        "algorithm": "safePrimes",
        "mode": "keyVer",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "safePrimeGroup": "ffdhe2048",
                "testType": "AFT",
                "tests": [
                    {
                        "tcId": 1,
                        "x": "399C088E4A1E1A03...",
                        "y": "FADA8667E9126779..."
                    },
                    {
                        "tcId": 2,
                        "x": "1DB3138EF400DDA7...",
                        "y": "70AACAB9A69AFE62..."
                    },
```

```
                    {
                        "tcId": 3,
                        "x": "099B19789CF2239F...",
                        "y": "C8F7038CB275E50F..."
                    }
                ]
            },
            {
                "tgId": 2,
                "testType": "AFT",
                "safePrimeGroup": "MODP-2048",
                "tests": [
                    {
                        "tcId": 4,
                        "x": "248091D90CB00F58EF...",
                        "y": "C08BF18980879C066E..."
                    },
                    {
                        "tcId": 5,
                        "x": "74B61CE8B689BAA23B...",
                        "y": "8EE385FFA770C9C9BF..."
                    },
                    {
                        "tcId": 6,
                        "x": "1B73FF75B0A20D99D0...",
                        "y": "4BF4509C0258E8E484..."
                    }
                ]
            }
        ]
    }
]
```


Figure 6

7. Test Vector Responses

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

Table 10 — Response JSON Object

JSON Property	Description	JSON Type
acvVersion	The ACVP version used	string
vsId	The vector set identifier	integer
testGroups	The test group objects in the response, see Table 11	array

An example of this is the following

```
{
    "acvVersion": "version",
    "vsId": 1,
    "testGroups": [ ... ]
}
```

Figure 7

The 'testGroups' section is used to organize the ACVP client response in a similar manner to how it distributes vectors. Some algorithm / mode / revision combinations might require that additional test group properties are provided in the response.

Table 11 — Response Group Objects

JSON Property	Description	JSON Type
tgId	The test group identifier	integer
tests	The test case objects in the response, depending on the algorithm see Table 13 or Table 15	array

An example of this is the following

```
{
    "tgId": 1,
    "tests": [ ... ]
}
```

Figure 8

7.1. Safe Primes keyGen Test Vector Responses

The test groups for SafePrimes / keyGen / 1.0 contain public key properties. The groups can be described using the following table.

Table 12 — Safe Primes keyGen Test Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
tests	The individual test cases for the group	array

Each test group contains an array of one or more test cases. Each test case is a JSON object that represents a single test vector to be processed by the ACVP client. The following table describes the JSON elements for each safePrimes / keyGen / 1.0 test vector.

Table 13 — Safe Primes keyGen Test Case Response JSON Object

JSON Value	Description	JSON type
tcId	The test case identifier	integer
x	The private key component X	hex
y	The public key component Y	hex

The following is an example JSON test vector response object for safePrimes / keyGen / 1.0.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 0,
        "testGroups": [
          {
            "tgId": 1,
            "tests": [
              {
                "tcId": 1,
                "x": "6316A9021906CB3F9F6...",
                "y": "8520DE9F113D659F708..."
              }
            ]
          }
        ]
    }
]
```

Figure 9

7.2. Safe Primes keyVer Test Vector Responses

The test groups for SafePrimes / keyVer / 1.0 contain public key properties. The groups can be described using the following table.

Table 14 — Safe Primes keyVer Test Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
tests	The individual test cases for the group	array

Each test group contains an array of one or more test cases. Each test case is a JSON object that represents a single test vector to be processed by the ACVP client. The following table describes the JSON elements for each SafePrimes / keyVer / 1.0 test vector.

Table 15 — Safe Primes keyVer Test Case Response JSON Object

JSON Value	Description	JSON type
tcId	The test case identifier	integer
testPassed	Whether or not the key verified	boolean

The following is an example JSON test vector response object for SafePrimes / keyVer / 1.0.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 0,
        "testGroups": [
          {
            "tgId": 1,
            "tests": [
              {
                "tcId": 1,
                "testPassed": true
              }
            ]
          }
        ]
    }
]
```

Figure 10

8. Security Considerations

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

Appendix A — Terminology

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

A.1.

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

Appendix B — Abbreviations and Acronyms

ACVP	Automated Crypto Validation Protocol
JSON	Javascript Object Notation

Appendix C — Revision History

Table C-1

Version	Release Date	Updates
1	2019-06-05	Initial Release

Appendix D — References

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

T. Kivinen, M. Kojo (May 2003) *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)* (Internet Engineering Task Force), RFC 3526, May 2003. RFC 3526. RFC RFC3526. DOI 10.17487/RFC3526. https://www.rfc-editor.org/info/rfc3526.

D. Gillmor (August 2016) *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)* (Internet Engineering Task Force), RFC 7919, August 2016. RFC 7919. RFC RFC7919. DOI 10.17487/RFC7919. https://www.rfc-editor.org/info/rfc7919.

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

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

National Institute of Standards and Technology (July 2013) *Digital Signature Standard (DSS)* (Gaithersburg, MD), July 2013. FIPS 186-4. https://doi.org/10.6028/NIST.FIPS.186-4.

Elaine B. Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, Richard Davis (April 2018) *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography* (Gaithersburg, MD), April 2018. SP 800-56A Rev. 3. https://doi.org/10.6028/NIST.SP.800-56Ar3.

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.