ACVP DSA Algorithm JSON Specification

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

June 01, 2016

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

Abstract

This document defines the JSON schema for testing FIPS 186-4 DSA 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 FIPS 186-4 DSA 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 FIPS 186-4 DSA implementations using ACVP.

2. Supported DSA Algorithms

The following DSA algorithm / mode / revision combinations **MAY** be advertised by the ACVP compliant cryptographic module:

- DSA / keyGen / 1.0
- DSA / pqgGen / 1.0
- DSA / pqgVer / 1.0
- DSA / sigGen / 1.0
- DSA / sigVer / 1.0

2.1. Supported Conformances for DSA Algorithms

The following DSA algorithms **MAY** claim conformance to [SP 800-106]:

- DSA / sigGen / 1.0
- DSA / sigVer / 1.0

3. Test Types and Test Coverage

3.1. Test Types

The ACVP server performs a set of tests on the specified DSA 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 DSA capabilities. This section describes the design of the tests used to validate implementations of the DSA algorithms.

- DSA / keyGen / 1.0 "AFT" — Algorithm Functional Test. The IUT is REQUIRED for each test case provided, to generate a key pair based on a generated group level PQG. This information is then communicated to the ACVP server and validated.
- DSA / pqgGen / 1.0 "GDT" — Generated Data Test. The IUT in this test mode is **REQUIRED** to generate PQ or G as a response to the ACVP provided test vector set.
- DSA / pqgVer / 1.0 "GDT" — Generated Data Test. In this test mode, the ACVP server is **REQUIRED** to generate domain parameters for transmission to the IUT. The IUT is expected to evaluate the validity of the domain parameters.
- DSA / sigGen / 1.0 "AFT" — Algorithm Functional Test. This testing mode expects the IUT to generate valid signatures based on the ACVP provided message. The IUT communicates the PQG, public key, and signature to the ACVP server, and the signature is then validated.
- DSA / sigVer / 1.0 "AFT" — Algorithm Functional Test. The ACVP server generates a series of signatures to communicate to the IUT. The IUT is **REQUIRED** to determine the validity of the signature given the PQG, key, and message.

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [FIPS 186-4] and [SP 800-89].

3.2.1. Requirements Covered

- FIPS186-4 Section 3 General Discussion. Domain parameter generation, key generation, signature generation, and signature validation are all within scope of ACVP server testing.
- FIPS186-4 Section 4 The Digital Signature Algorithm (DSA). The ACVP server provides a means of the generation and validation of domain parameters. The ACVP server is **SHALL** support a variety of parameter sizes/hash function for creation and delivery to/from the IUT. The ACVP server **SHALL** allow for the testing of the validity of domain parameters. Key pair generation testing **SHALL** be provided by the ACVP server. Both Signature Generation and Validation testing mechanmisms **SHALL** be provided by the ACVP server.
- SP800-106 Section 3 Randomized Hashing and Section 4 Digital Signatures Using Randomized Hashing. The IUT **SHALL** be provided or provide a random value that should be used to "randomize" a message prior to signing and/or verifying an original message.

3.2.2. Requirements Not Covered

- FIPS186-4 Section 3 General Discussion. Assurances of private key secrecy and ownership **SHALL NOT** be within scope of ACVP testing.
- FIPS186-4 Section 4 The Digital Signature Algorithm (DSA). The IUT's selection of parameter sizes and hash functions **SHALL NOT** be within scope of ACVP server testing. Though the ACVP server **SHALL** support a variety of parameter sizes/hash functions, the IUT's selection of these is out of scope of testing. The ACVP server **MAY** provide testing for the validity of domain parameters, but testing **SHALL NOT** provide assurances the IUT has validated a set of domain parameters prior to their use. Domain parameter and key pair management **SHALL NOT** be within scope of ACVP testing.
- SP800-106 Section 3.3 The Random Value. DSA, ECDSA, and RSA have random values generated as per their signing process, this random value can be used as the input to the message randomization function, doing so however is out of scope of this testing.

4. Capabilities Registration

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

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

4.1. Prerequisites

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

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

Table 1 — Prerequisite Properties

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

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

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

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

```
]
```

Figure 1

4.2. Required Prerequisite Algorithms for DSA Validations

Each DSA implementation relies on other cryptographic primitives. For example, DSA 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 DSA 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. DSA Algorithm Capabilities Registration

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

Table 3 — DSA Algorithm Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	The algorithm under test	string	"DSA"
mode	The DSA mode to be validated	string	"pqgGen", "pqgVer", "keyGen", "sigGen", "sigVer"
revision	The algorithm testing revision to use.	string	"1.0"
prereqVals	prerequistie algorithm validations	array of prereqAlgVal objects	See Section 4.2
capabilities	array of JSON objects, each with fields pertaining to the global DSA mode indicated above and identified uniquely	Array of JSON objects	See Section 4.4, Section 4.5, Section 4.5.1, Section 4.5.2 or Section 4.5.3

JSON Value	Description	JSON type	Valid Values
	by the combination of the DSA "mode" and indicated properties		
conformances	Used to denote the optional conformances that can apply to specific modes of DSA.	Array of strings	See Section 2.1

4.4. DSA pqgGen Mode Capabilities

The DSA / pqgGen / 1.0 mode capabilities are advertised as JSON objects, which are elements of the 'capabilities' array in the ACVP registration message. See the ACVP specification for details on the registration message.

Each DSA / pqgGen / 1.0 mode capability set is advertised as a self-contained JSON object.

The complete list of DSA PQ and G generation capabilities may be advertised by the ACVP compliant crypto module:

Table 4 — DSA pqgGen Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
l	The length in bits of the field and the length in bits of prime p.	value	2048, or 3072
n	The length in bits of q which is a prime factor of (p-1).	value	224, or 256 when L is 2048. Otherwise 256.
pqGen	The methods supported to generate p and q.	array	Any non-empty subset of {"probable", "provable"}.
gGen	The methods supported to generate g.	array	Any non-empty subset of {"unverifiable", "canonical"}.
hashAlg	The hash functions supported when generating p, q and g.	array	Any non-empty subset of {"SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}. Note that the digest size of the hash function **MUST** be equal to or greater than N.

An example of the DSA / pqgGen / 1.0 registration is the following

```
{
  "algorithm": "DSA",
  "mode": "pqgGen",
  "revision": "1.0",
  "prereqVals": [
```

```
    {
      "algorithm": "SHA",
      "valValue": "123456"
    },
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    }
  ],
  "capabilities": [
    {
      "pqGen": [
        "probable",
        "provable"
      ],
      "gGen": [
        "unverifiable",
        "canonical"
      ],
      "l": 2048,
      "n": 224,
      "hashAlg": [
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    },
    {
      "pqGen": [
        "probable",
        "provable"
      ],
      "gGen": [
        "unverifiable",
        "canonical"
      ],
      "l": 2048,
      "n": 256,
      "hashAlg": [
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/256"
```

```
      ]
    },
    {
      "pqGen": [
        "probable",
        "provable"
      ],
      "gGen": [
        "unverifiable",
        "canonical"
      ],
      "l": 3072,
      "n": 256,
      "hashAlg": [
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/256"
      ]
    }
  ]
}
```

Figure 2

4.5. DSA pqgVer Mode Capabilities

The DSA / pqgVer / 1.0 mode capabilities are advertised as JSON objects, which are elements of the 'capabilities' array in the ACVP registration message. See the ACVP specification for details on the registration message.

Each DSA / pqgVer / 1.0 mode capability set is advertised as a self-contained JSON object.

The complete list of DSA P, Q and G verification capabilities may be advertised by the ACVP compliant crypto module:

Table 5 — DSA pqgVer Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
l	The length in bits of the field and the length in bits of prime p.	value	1024, 2048, or 3072
n	The length in bits of q which is a prime factor of (p-1).	value	160 when L is 1024. 224, or 256 when L is 2048. Otherwise 256.
pqGen	The methods supported to generate p and q.	array	Any non-empty subset of {"probable", "provable"}.

JSON Value	Description	JSON type	Valid Values
gGen	The methods supported to generate g.	array	Any non-empty subset of {"unverifiable", "canonical"}.
hashAlg	The hash functions supported when generating p, q and g.	array	Any non-empty subset of {"SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}. Note that the digest size of the hash function MUST be equal to or greater than N.

An example of the DSA / pqgVer / 1.0 registration is the following

```
{
  "algorithm": "DSA",
  "mode": "pqgVer",
  "revision": "1.0",
  "prereqVals": [
    {
      "algorithm": "SHA",
      "valValue": "123456"
    },
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    }
  ],
  "capabilities": [
    {
      "pqGen": [
        "probable",
        "provable"
      ],
      "gGen": [
        "unverifiable",
        "canonical"
      ],
      "l": 1024,
      "n": 160,
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
```

```
      ]
    },
    {
      "pqGen": [
        "probable",
        "provable"
      ],
      "gGen": [
        "unverifiable",
        "canonical"
      ],
      "l": 2048,
      "n": 224,
      "hashAlg": [
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    },
    {
      "pqGen": [
        "probable",
        "provable"
      ],
      "gGen": [
        "unverifiable",
        "canonical"
      ],
      "l": 2048,
      "n": 256,
      "hashAlg": [
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/256"
      ]
    },
    {
      "pqGen": [
        "probable",
        "provable"
      ],
      "gGen": [
```

```
        "unverifiable",
        "canonical"
      ],
      "l": 3072,
      "n": 256,
      "hashAlg": [
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/256"
      ]
    }
  ]
}
```

Figure 3

4.5.1. DSA keyGen Mode Capabilities

The DSA / keyGen / 1.0 mode capabilities are advertised as JSON objects, which are elements of the 'capabilities' array in the ACVP registration message. See the ACVP specification for details on the registration message.

Each DSA / keyGen / 1.0 mode capability set is advertised as a self-contained JSON object.

The complete list of DSA key generation capabilities may be advertised by the ACVP compliant crypto module:

Table 6 — DSA keyGen Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
l	The length in bits of the field and the length in bits of prime p.	value	2048, or 3072
n	The length in bits of q which is a prime factor of (p-1).	value	224, or 256 when L is 2048. Otherwise 256.

An example of this is the following

```
{
  "algorithm": "DSA",
  "mode": "keyGen",
  "revision": "1.0",
  "prereqVals": [
    {
      "algorithm": "SHA",
      "valValue": "123456"
    },
    {
```

```
      "algorithm": "DRBG",
      "valValue": "123456"
    }
  ],
  "capabilities": [
    {
      "n": 224,
      "l": 2048
    },
    {
      "n": 256,
      "l": 2048
    },
    {
      "n": 256,
      "l": 3072
    }
  ]
}
```

Figure 4

4.5.2. DSA sigGen Mode Capabilities

The DSA / sigGen / 1.0 mode capabilities are advertised as JSON objects, which are elements of the 'capabilities' array in the ACVP registration message. See the ACVP specification for details on the registration message.

Each DSA / sigGen / 1.0 mode capability set is advertised as a self-contained JSON object.

The complete list of DSA signature generation capabilities may be advertised by the ACVP compliant crypto module:

Table 7 — DSA sigGen Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
l	The length in bits of the field and the length in bits of prime p.	value	2048, or 3072
n	The length in bits of q which is a prime factor of (p-1).	value	224, or 256 when L is 2048. Otherwise 256.
hashAlg	The hash functions supported when signing a message.	array	Any non-empty subset of {"SHA2-224","SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}.

An example of this is the following


```
{
  "algorithm": "DSA",
  "mode": "sigGen",
  "revision": "1.0",
  "prereqVals": [
    {
      "algorithm": "SHA",
      "valValue": "123456"
    },
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    }
  ],
  "capabilities": [
    {
      "n": 224,
      "l": 2048,
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    },
    {
      "n": 256,
      "l": 2048,
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    },
    {
      "n": 256,
      "l": 3072,
```

```
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    }
  ],
  "conformances": [
    "SP800-106"
  ]
}
```

Figure 5

4.5.3. DSA sigVer Mode Capabilities

The DSA / sigVer / 1.0 mode capabilities are advertised as JSON objects, which are elements of the 'capabilities' array in the ACVP registration message. See the ACVP specification for details on the registration message.

Each DSA / sigVer / 1.0 mode capability set is advertised as a self-contained JSON object.

The complete list of DSA signature verification capabilities may be advertised by the ACVP compliant crypto module:

Table 8 — DSA sigVer Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
l	The length in bits of the field and the length in bits of prime p.	value	1024, 2048, or 3072
n	The length in bits of q which is a prime factor of (p-1).	value	160 when L is 1024. 224, or 256 when L is2048. Otherwise 256.
hashAlg	The hash functions supported when verifying a message.	array	Any non-empty subset of {"SHA-1","SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}.

An example of this is the following

```
{
```

```
  "algorithm": "DSA",
  "mode": "sigVer",
  "revision": "1.0",
  "prereqVals": [
    {
      "algorithm": "SHA",
      "valValue": "123456"
    },
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    }
  ],
  "capabilities": [
    {
      "n": 160,
      "l": 1024,
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    },
    {
      "n": 224,
      "l": 2048,
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    },
    {
      "n": 256,
      "l": 2048,
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
```

```
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    },
    {
      "n": 256,
      "l": 3072,
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    }
  ],
  "conformances": [
    "SP800-106"
  ]
}
```

Figure 6

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 FIPS 186-4 DSA 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 9 — 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 7

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 DSA / sigGen / 1.0 etc. This section describes the JSON schema for a test vector set used with DSA 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 10 — DSA 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 6.1.1, Section 6.2.1, Section 6.3.1, Section 6.4.1, or Section 6.5.1 depending on the algorithm	array

6.1. DSA pqgGen Test Vectors

6.1.1. DSA pqgGen 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 DSA / pqgGen / * is as follows:

Table 11 — DSA PQGGen Test Group JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
testType	The test operation performed	string
l	Length in bits of prime modulus p	integer
n	Length in bits of prime divisor q	integer

JSON Value	Description	JSON type
pqMode	The specific pq generation mode used in the test group	string
gMode	The specific g generation mode used in the test group	string
hashAlg	The hash algorithm used in the test group	string
tests	Array of individual test vector JSON objects, which are defined in Section 6.1.2	array

6.1.2. DSA pqgGen 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 DSA / pqgGen / 1.0 test vector.

Table 12 — DSA PQGGen Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
p	The prime modulus	hex
q	The prime divisor of p — 1	hex
domainSeed	The seed used to generate p and q in the probable method	hex
index	The index value provided to the g generator in the canonical method	hex

NOTE 1 – For groups generating a p and q value, only the 'tcId' property will be present. It is the client's responsibility to generate a valid p and q pair with the specified properties using the specified generation method from [FIPS 186-4].

NOTE 2 – For groups generating a g value, using the "unverifiable" method, only the 'tcId', 'p' and 'q' will be provided. For groups generating a g value using the "canonical" method, an additional 'domainSeed' and 'index' will be provided. For more information about these generation methods see Appendix A in [FIPS 186-4].

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

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "DSA",
```

```
"mode": "pqgGen",
"revision": "1.0",
"testGroups": [
    {
        "tgId": 1,
        "l": 2048,
        "n": 224,
        "hashAlg": "SHA2-224",
        "pqMode": "probable",
        "testType": "GDT",
        "tests": [
            {
                "tcId": 1,
            }
        ]
    },
    {
        "tgId": 2,
        "l": 2048,
        "n": 224,
        "hashAlg": "SHA2-384",
        "pqMode": "provable",
        "testType": "GDT",
        "tests": [
            {
                "tcId": 2,
            }
        ]
    },
    {
        "tgId": 3,
        "l": 2048,
        "n": 224,
        "hashAlg": "SHA2-224",
        "testType": "GDT",
        "gMode": "unverifiable",
        "tests": [
            {
              "tcId": 3,
              "p": "B9D5DEC1F8541708F...",
              "q": "9F3FCC1DA20ACCD5C..."
            }
        ]
    },
    {
        "tgId": 4,
```

```
                "l": 2048,
                "n": 224,
                "hashAlg": "SHA2-224",
                "testType": "GDT",
                "gMode": "canonical",
                "tests": [
                    {
                      "tcId": 4,
                      "p": "CACDDA5F26C38B7EF...",
                      "q": "A4D538BAE42A35316...",
                      "domainSeed": "E8A171F4...",
                      "index": "AD"
                    }
                ]
            }
        ]
    }
]
```

Figure 8

6.2. DSA pqgVer Test Vectors

6.2.1. DSA pqgVer 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 DSA / pqgVer / * is as follows:

Table 13 — DSA PQGVer Test Group JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
testType	The test operation performed	string
l	Length in bits of prime modulus p	integer
n	Length in bits of prime divisor q	integer
pqMode	The specific pq generation mode used in the test group	string
gMode	The specific g generation mode used in the test group	string
hashAlg	The hash algorithm used in the test group	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. DSA pqgVer 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 DSA / pqgVer / 1.0 test vector.

Table 14 — DSA PQGVer Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
p	The prime modulus	hex
q	The prime divisor of p — 1	hex
domainSeed	The seed used to generate p and q in the probable method	hex
counter	The counter used to generate p and q in the probable method	integer
pSeed	The seed used to generate p in the provable method	hex
qSeed	The seed used to generate q in the provable method	hex
pCounter	The counter used to generate p in the provable method	integer
qCounter	The counter used to generate q in the provable method	integer
g	The generator	hex
h	The index value provided to the g generator in the unverifiable method	hex
index	The index value provided to the g generator in the canonical method	hex

NOTE 1 – For groups verifying a p and q value using the “probable” method, only the ‘tcId’, ‘p’, ‘q’, ‘domainSeed’ and ‘counter’ properties will be present.

NOTE 2 – For groups verifying a p and q value using the “provable” method, only the ‘tcId’, ‘p’, ‘q’, ‘pCounter’, ‘qCounter’, ‘pSeed’ and ‘qSeed’ properties will be present.

NOTE 3 – For groups verifying a g value using the “unverifiable” method, only the ‘tcId’, ‘p’, ‘q’, ‘g’, ‘h’, and ‘domainSeed’ properties will be present.

NOTE 4 – For groups verifying a g value using the "canonical" method, only the 'tcId', 'p', 'q', 'g', and 'index' properties will be present.

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

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "DSA",
        "mode": "pqgVer",
        "revision": "1.0",
        "testGroups": [
          {
            "tgId": 1,
            "l": 1024,
            "n": 160,
            "hashAlg": "SHA-1",
            "testType": "GDT",
            "pqMode": "probable",
            "tests": [
              {
                "tcId": 1,
                "p": "9D9269AC94DB5003355...",
                "q": "C5C97FD66D441234E78...",
                "domainSeed": "259947680F...",
                "counter": 309
              },
            ]
          },
          {
            "tgId": 2,
            "l": 1024,
            "n": 160,
            "hashAlg": "SHA-1",
            "testType": "GDT",
            "pqMode": "provable",
            "tests": [
              {
                "tcId": 36,
                "p": "4BDC98F8302E24CEDCE...",
                "q": "A00C3FEAF8F56910DA5...",
```

```
            "pCounter": 1874,
            "qCounter": 115,
            "pSeed": "C4967615C4E1391...",
            "qSeed": "C4967615C4E1391..."
          },
        ]
      },
      {
        "tgId": 3,
        "l": 1024,
        "n": 160,
        "hashAlg": "SHA-1",
        "testType": "GDT",
        "gMode": "unverifiable",
        "tests": [
          {
            "tcId": 211,
            "p": "A1648B0F29F5D38DA507...",
            "q": "9BCF7E1625844A88EABB...",
            "g": "0FB0987B157E12F15D78...",
            "h": "02",
            "domainSeed": "D9F63E102A9..."
          },
        ]
      },
      {
        "tgId": 4,
        "l": 1024,
        "n": 160,
        "hashAlg": "SHA-1",
        "testType": "GDT",
        "gMode": "canonical",
        "tests": [
          {
            "tcId": 246,
            "p": "9A1B46A4498962D12FDE...",
            "q": "B70E07662CADF2A41914...",
            "g": "45659A0B48B5B581E5CA...",
            "index": "45"
          },
        ]
      }
    ]
}
```

```
]
```

Figure 9

6.3. DSA keyGen Test Vectors

6.3.1. DSA 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 DSA / keyGen / * is as follows:

Table 15 — DSA keyGen Test Group JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
testType	The test operation performed	string
l	Length in bits of prime modulus p	integer
n	Length in bits of prime divisor q	integer
tests	Array of individual test vector JSON objects, which are defined in Section 6.3.2	array

6.3.2. DSA 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 DSA / keyGen / 1.0 test vector.

Table 16 — DSA keyGen Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
NOTE – The client is responsible for generating domain parameters and a key to be verified by the server.		

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

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

```
        "vsId": 1564,
        "algorithm": "DSA",
        "mode": "keyGen",
        "revision": "1.0",
        "testGroups": [
          {
            "tgId": 1,
            "l": 2048,
            "n": 224,
            "tests": [
              {
                "tcId": 1
              },
            ]
          }
        ]
    }
]
```

Figure 10

6.4. DSA sigGen Test Vectors

6.4.1. DSA sigGen 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 DSA / sigGen / * is as follows:

Table 17 — DSA sigGen Test Group JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
testType	The test operation performed	string
l	Length in bits of prime modulus p	integer
n	Length in bits of prime divisor q	integer
hashAlg	The hash algorithm used in the test group	string
conformance	Signifies all test cases within the group should utilize random message hashing as described in [SP 800-106].	string

JSON Value	Description	JSON type
tests	Array of individual test vector JSON objects, which are defined in Section 6.4.2	array

6.4.2. DSA sigGen 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 DSA / sigGen / 1.0 test vector.

Table 18 — DSA sigGen Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
message	The message used to generate signature or verify signature	hex

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

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "DSA",
        "mode": "sigGen",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "l": 2048,
                "n": 224,
                "hashAlg": "SHA-1",
                "tests": [
                    {
                        "tcId": 1,
                        "message": "C1FEBB069145F..."
                    }
                ]
            },
            {
                "tgId": 2,
                "l": 2048,
```

```
                "n": 224,
                "hashAlg": "SHA-1",
                "conformance": "SP800-106",
                "tests": [
                    {
                        "tcId": 2,
                        "message": "C1FEBB069145F..."
                    }
                ]
            }
        ]
    }
]
```

Figure 11

6.5. DSA sigVer Test Vectors

6.5.1. DSA sigVer 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 DSA / sigVer / * is as follows:

Table 19 — DSA sigVer Test Group JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
testType	The test operation performed	string
l	Length in bits of prime modulus p	integer
n	Length in bits of prime divisor q	integer
p	Domain parameter P	hex
q	Domain parameter Q	hex
g	Domain parameter G	hex
hashAlg	The hash algorithm used in the test group	string
conformance	Signifies all test cases within the group should utilize random message hashing as described in [SP 800-106].	string
tests	Array of individual test vector JSON objects, which are defined in Section 6.5.2	array

6.5.2. DSA sigVer 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 DSA / sigVer / 1.0 test vector.

Table 20 — DSA sigVer Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
message	The message used to generate signature or verify signature	hex
randomValue	The random value to be used as an input into the message randomization function as described in [SP 800-106].	hex
randomValueLen	The random value's bit length.	integer
r	The signature component R	hex
s	The signature component S	hex
y	The public key component Y	hex
NOTE – The 'randomValue' and 'randomValueLen' properties are only present if the 'conformance' property of the group is set to "SP800-106".		

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

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "DSA",
        "mode": "sigVer",
        "revision": "1.0",
        "testGroups": [
          {
            "tgId": 1,
            "l": 1024,
            "n": 160,
            "p": "A48828B4A4E149C2D1FC66F108D370A2A9E87...",
            "q": "D3C53E62338D4231E42FA9683175C404FBF52...",
            "g": "7BDDD6B8E9B4667397B278F98F446C418EF1A...",
            "hashAlg": "SHA-1",
            "tests": [
```

```
            {
              "tcId": 1,
              "message": "1A128C1A61091CE52A8B89D69A3...",
              "y": "A3A2F1AFC3091204B7D7ED3617A80E0FD...",
              "r": "C390851DE10669399308D9F401B0286AA...",
              "s": "26BE6EFDCD2ED6946BFCFE2D396AB3A2E..."
            },
          ]
        },
        {
          "tgId": 2,
          "l": 1024,
          "n": 160,
          "p": "66F108D370A2A9E87DBD49BC09A27017621A...",
          "q": "1F855467465A653D2245B7E31C910A18EF79...",
          "g": "F098ACF73BFC43B1B5E7BFB065FBBDDD6B8E...",
          "hashAlg": "SHA-1",
          "conformance": "SP800-106",
          "tests": [
            {
              "tcId": 2,
              "message": "A0F409909E31C2C23BB7A24B8103F11F...",
              "randomValue": "DB0D25A72C8DEFA0F409909E31C2..."
              "randomValueLen": 1024,
              "y": "A3A284CB56F3A0EF37749B5EA4D2EC824ED5E7EAD481BE520B...",
              "r": "15FE6F6DAEFD42CD5FA7444221CB545BB0CDB95DC9D76E0...",
              "s": "B9AF193E6832934537B0B05D177BA8F7DA48D9DD84D27B8AFE9..."
            },
          ]
        }
      ]
    }
]
```


Figure 12

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 21 — 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 22	array

An example of this is the following

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

Figure 13

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 22 — 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 24, Table 26, Table 28, Table 30 or Table 32	array

An example of this is the following

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

Figure 14

7.1. DSA PQGGen Test Vector Responses

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

Table 23 — DSA PQGGen 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 DSA / pqgGen / 1.0 test vector.

Table 24 — DSA PQGGen Test Case Response JSON Object

JSON Value	Description	JSON type
tcId	The test case identifier	integer
p	The prime modulus	hex
q	The prime divisor of p — 1	hex
g	The generator	hex
domainSeed	The seed used to generate p and q in the probable method	hex
counter	The counter used to generate p and q in the probable method	integer
pSeed	The seed used to generate p in the provable method	hex
qSeed	The seed used to generate q in the provable method	hex
pCounter	The counter used to generate p in the provable method	integer
qCounter	The counter used to generate q in the provable method	integer
NOTE 1 – The properties ‘p’, ‘q’, ‘domainSeed’ and ‘counter’ are only required in test groups generating p and q using the probable generation method in [FIPS 186-4]. An example of this is ‘tgId’ 1 in the following example. NOTE 2 – The properties ‘p’, ‘q’, ‘domainSeed’, ‘pSeed’, ‘qSeed’, ‘pCounter’ and ‘qCounter’ are only required in test groups generating p and q using the provable generation method in [FIPS 186-4]. An example of this is ‘tgId’ 2 in the following example. NOTE 3 – The property ‘g’ is only required in test groups generating g under both allowed modes of generation, canonical and unverifiable. An example of this is ‘tgId’ 3 and 4 in the following example.		

The following is an example JSON test vector response object for DSA / pqgGen / 1.0.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "testGroups": [
            {
              "tgId": 1,
              "tests": [
                {
                  "tcId": 1,
                  "p":"E0BB55A249993FE4...",
                  "q":"C0074BDDC42F22F5...",
                  "domainSeed":"01AA98A...",
                  "counter":379
                }
              ]
            },
            {
              "tgId": 2,
              "tests": [
                {
                  "tcId": 2,
                  "p":"84B73C1CE8E8C10F8...",
                  "q":"A0B7917C9020F2332...",
                  "domainSeed":"98179EF2...",
                  "pSeed":"98179EF2D7FD0...",
                  "qSeed":"98179EF2D7FD0...",
                  "pCounter":596,
                  "qCounter":255
                }
              ]
            },
            {
              "tgId": 3,
              "tests": [
                {
                  "tcId": 3,
                  "g": "01098AD5E87869EF..."
                }
              ]
            },
            {
              "tgId": 4,
              "tests": [
```

```
                {
                  "tcId": 4,
                  "g": "5C4AB5D4C901A375..."
                }
              ]
            }
        ]
    }
]
```

Figure 15

7.2. DSA PQGVer Test Vector Responses

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

Table 25 — DSA PQGVer 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 DSA / pqgVer / 1.0 test vector.

Table 26 — DSA PQGVer Test Case Response JSON Object

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

The following is an example JSON test vector response object for DSA / pqgVer / 1.0.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "testGroups": [
          {
            "tgId": 1,
            "tests": [
              {
                "tcId": 1,
```

```
            "testPassed": false,
          }
        ]
      },
      {
        "tgId": 2,
        "tests": [
          {
            "tcId": 36,
            "testPassed": false,
          }
        ]
      },
      {
        "tgId": 3,
        "tests": [
          {
            "tcId": 211,
            "testPassed": true,
          }
        ]
      },
      {
        "tgId": 4,
        "tests": [
          {
            "tcId": 246,
            "testPassed": true,
          }
        ]
      }
    ]
  }
]
```


Figure 16

7.3. DSA keyGen Test Vector Responses

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

Table 27 — DSA keyGen Test Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
p	The prime modulus	hex

JSON Value	Description	JSON type
q	The prime divisor of p — 1	hex
g	The generator	hex
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 DSA / keyGen / 1.0 test vector.

Table 28 — DSA 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 DSA / keyGen / 1.0.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "testGroups": [
          {
            "tgId": 1,
            "p": "9D75F12AD16C09A618F3D25...",
            "q": "F9146BFEC592547B8C69737...",
            "g": "96A7DD911D076093EBBA4D9...",
            "tests": [
              {
                "tcId": 1,
                "x": "6316A9021906CB3F9F6...",
                "y": "8520DE9F113D659F708..."
              }
            ]
          }
        ]
    }
]
```

Figure 17

7.4. DSA sigGen Test Vector Responses

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

Table 29 — DSA sigGen Test Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
p	The prime modulus	hex
q	The prime divisor of p — 1	hex
g	The generator	hex
y	The public key component Y	hex
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 DSA / sigGen / 1.0 test vector.

Table 30 — DSA sigGen Test Case Response JSON Object

JSON Value	Description	JSON type
tcId	The test case identifier	integer
r	The signature component R	hex
s	The signature component S	hex
randomValue	The random value to be used as an input into the message randomization function as described in [SP 800-106].	hex
randomValueLen	The random value's bit length.	integer
NOTE – The properties 'randomValue' and 'randomValueLen' are only required for groups that had the 'conformance' property set to "SP800-106".		

The following is an example JSON test vector response object for DSA / sigGen / 1.0.

```
[
  {
    "acvVersion": <acv-version>
  },
  {
    "vsId": 1564,
    "testGroups": [
      {
        "tgId": 1,
        "p": "C8DC6C77CC31ADAE68F3221D59C7...",
        "q": "B1B7BCF2467F8CC46FCDD2A41327...",
```

```
      "g": "6C07130A6F6D05AAC9350936EE06...",
      "y": "79143F63ECCC06B3D35C61DA2FB8...",
      "tests": [
        {
          "tcId": 1,
          "r": "4E7F70A92EC0E6871E2EA278...",
          "s": "641A74C54A0B642DED9FE6EE..."
        }
      ]
    },
    {
      "tgId": 2,
      "p": "77CC31ADAE68F3221D59C75538F6...",
      "q": "82E17546E564816AFB54987C879A...",
      "g": "B1303035FE460CEBF21DDA00CC08...",
      "y": "D699C3B6E150E60443BA461C2564...",
      "tests": [
        {
          "tcId": 2,
          "r": "4E7F70A92EC0E6871E2EA275...",
          "s": "641A74C54A0B642DED9FE6EE...",
          "randomValue": "23678643ACB728...",
          "randomValueLen": 1024
        }
      ]
    }
  ]
 }
]
```


Figure 18

7.5. DSA sigVer Test Vector Responses

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

Table 31 — DSA sigVer 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 DSA / sigVer / 1.0 test vector.

Table 32 — DSA sigVer Test Case Response JSON Object

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

The following is an example JSON test vector response object for DSA / sigVer / 1.0.

```
[
  {
    "acvVersion": <acvp-version>
  },
  {
    "vsId": 1564,
    "testGroups": [
      {
        "tgId": 1,
        "tests": [
          {
            "tcId": 1,
            "testPassed": true
          },
          {
            "tcId": 2,
            "testPassed": true
          }
        ]
      }
    ]
  }
]
```

Figure 19

8. Security Considerations

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

Unresolved directive in draft-fussell-acvp-dsa.adoc — include::/__w/ACVP-standalone/ACVP-standalone/src/common/common-sections/99-acknowledgements.adoc[]

Appendix A — References

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

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 (November 2006) *Recommendation for Obtaining Assurances for Digital Signature Applications* (Gaithersburg, MD), November 2006. SP 800-89. https://doi.org/10.6028/NIST.SP.800-89.

Quynh H. Dang (February 2009) *Randomized Hashing for Digital Signatures* (Gaithersburg, MD), February 2009. SP 800-106. https://doi.org/10.6028/NIST.SP.800-106.

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.