# ACVP EdDSA Algorithm JSON Specification

Christopher Celi
*Information Technology Laboratory*
*Computer Security Division*

August 01, 2018

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

## Abstract

This document defines the JSON schema for testing FIPS Pub 186 EdDSA 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 Pub 186 EdDSA 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 Pub 186 EdDSA implementations using ACVP.

## 2. Supported EDDSA Algorithms

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

- EDDSA / keyGen / 1.0
- EDDSA / keyVer / 1.0
- EDDSA / sigGen / 1.0
- EDDSA / sigVer / 1.0

# 3. Test Types and Test Coverage

## 3.1. Test Types

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

- EDDSA / keyGen / 1.0 "AFT" — Algorithm Functional Test. The IUT is **REQUIRED** for each test case provided, to generate a key pair based on an approved curve. This information is then communicated to the ACVP server and validated.
- EDDSA / keyVer / 1.0 "AFT" — Algorithm Functional Test. The ACVP server is **REQUIRED** to generate a series of keys based on the IUT provided NIST curve(s). The keys generated by the server **MAY** or **MAY NOT** be valid, the IUT is **REQUIRED** to determine if the keys provided in the test cases are valid or invalid keys as they relate to the curve.
- EDDSA / sigGen / 1.0 "AFT" — Algorithm Functional Test. This testing mode expects the IUT to generate valid signatures based on the ACVP provided message. The signature is then validated with the ACVP server given the IUT's communicated curve, public key, and signature.
- EDDSA / sigGen / 1.0 "BFT" — Bit Flip Test. This testing mode produces a single message and flips individual bits in the message to ensure a client is able to produce distinct signatures for each message.
- EDDSA / 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 curve, key, and message.

## 3.2. Test Coverage

- TBD…

### 3.2.1. Requirements Covered

- TBD…

### 3.2.2. Requirements Not Covered

- TBD…

# 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 EdDSA 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 EDDSA Validations

Each EDDSA implementation relies on other cryptographic primitives. For example, EDDSA 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 EDDSA Prerequisite Algorithms JSON Values**

| JSON Value | Description | JSON type | Valid Values |
|---|---|---|---|
| algorithm | a prerequisite algorithm | string | DRBG, SHA |
| 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. EDDSA Algorithm Capabilities Registration

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

**Table 3 — EDDSA Algorithm Capabilities JSON Values**

| JSON Value | Description | JSON type | Valid Values |
|---|---|---|---|
| algorithm | The algorithm under test | string | “EDDSA” |
| mode | The EDDSA mode to be validated | string | “keyGen”, “keyVer”, “sigGen”, or “sigVer” |
| revision | The algorithm testing revision to use | string | “1.0” |
| prereqVals | prerequistie algorithm validations | array of prereqAlgVal objects | See Section 4.2 |

The follwing sections offer additional **REQUIRED** JSON properties for each algorithm / mode / revision.

#### 4.3.1. The EDDSA keyGen Mode Capabilities

Each EDDSA keyGen mode capability set is advertised as a self-contained JSON object.

#### 4.3.1.1. EDDSA keyGen Full Set of Capabilities

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

**Table 4 — EDDSA keyGen Capabilities JSON Values**

| JSON Value | Description | JSON type | Valid Values |
|---|---|---|---|
| curve | The curve names supported for the IUT in keyGen | array | Any non-empty subset of {"ED-25519", "ED-448"} |
| secretGenerationMode | The method used to generate the randomness incoporated in the key | array | Any non-empty subset of {"extra bits", "testing candidates"} |
| NOTE – More information on the 'secretGenerationMode' can be found in Appendix A.2 in [FIPS 186-5 (Draft)]. | | | |

An example of this is the following

```
{
    "algorithm": "EDDSA",
    "mode": "keyGen",
    "revision": "1.0",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "curve": [
        "ED-25519",
        "ED-448"
    ],
    "secretGenerationMode": [
        "extra bits",
        "testing candidates"
    ]
}
```

**Figure 2**

### 4.3.2. The EDDSA keyVer Mode Capabilities

Each EDDSA keyVer mode capability set is advertised as a self-contained JSON object.

#### 4.3.2.1. keyVer Full Set of Capabilities

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

**Table 5 — EDDSA keyVer Capabilities JSON Values**

| JSON Value | Description | JSON type | Valid Values |
|---|---|---|---|
| curve | The curve names supported for the IUT in keyVer | array | Any non-empty subset of {"ED-25519", "ED-448"} |

An example of this is the following

```
{
    "algorithm": "EDDSA",
    "mode": "keyVer",
    "revision": "1.0",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "curve": [
        "ED-25519",
        "ED-448"
    ]
}
```

**Figure 3**

### 4.3.3. The sigGen Mode Capabilities

Each EDDSA sigGen mode capability set is advertised as a self-contained JSON object.

#### 4.3.3.1. sigGen Full Set of Capabilities

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

**Table 6 — EDDSA sigGen Capabilities JSON Values**

| JSON Value | Description | JSON type | Valid Values |
|---|---|---|---|
| curve | The curve names supported for the IUT in sigGen | array | Any non-empty subset of {"ED-25519", "ED-448"} |
| pure | If the IUT supports normal 'pure' EdDSA signature generation functionality | bool | true/false |
| preHash | If the IUT supports Prehash EdDSA, i.e., HashEdDSA, signature generation functionality | bool | true/false |

The following is an example

```
{
    "algorithm": "EDDSA",
    "mode": "sigGen",
    "revision": "1.0",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "pure": true,
    "preHash": true,
    "curve": [
        "ED-25519",
        "ED-448"
    ]
}
```

**Figure 4**

### 4.3.4. The sigVer Mode Capabilities

Each EDDSA sigVer mode capability set is advertised as a self-contained JSON object.

#### 4.3.4.1. sigVer Full Set of Capabilities

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

**Table 7 — EDDSA sigVer Capabilities JSON Values**

| JSON Value | Description | JSON type | Valid Values |
|---|---|---|---|
| curve | The curve names supported for the IUT in sigGen | array | Any non-empty subset of {"ED-25519", "ED-448"} |
| pure | If the IUT supports normal 'pure' sigGen functionality | bool | true/false |
| preHash | If the IUT supports accepting a preHashed message to sign | bool | true/false |

The following is an example

```
{
    "algorithm": "EDDSA",
    "mode": "sigVer",
    "revision": "1.0",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "pure": true,
    "preHash": true,
    "curve": [
        "ED-25519",
        "ED-448"
    ]
}
```

**Figure 5**

# 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 session would require multiple test vector sets to be downloaded and processed by the ACVP client. Each test vector set represents an individual EdDSA function. This section describes the JSON schema for a test vector set used with EdDSA 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 8 — Vector Set JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| acvVersion | Protocol version identifier | string |
| vsId | Unique numeric identifier for the vector set | string |
| 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 group JSON objects, which are defined in Section 5.1, Section 5.3, Section 5.5, and Section 5.7 | array |

An example of this would look like this

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

**Figure 6**

## 5.1. EDDSA 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 JSON elements of the Test Group JSON object.

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

**Table 9 — EDDSA keyGen Test Group JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | The test group identifier | integer |
| curve | The curve type used for the test vectors | string |
| secretGenerationMode | The method of generating a secret used for key generation in the test vectors | string |
| testType | The testType for the group | string |
| tests | Array of individual test vector JSON objects, which are defined in Section 5.2 | array |

## 5.2. EDDSA 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 EDDSA test vector.

**Table 10 — 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 a single key per test case. | | |

The following is an example of a prompt for EDDSA / keyGen / 1.0

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "EDDSA",
        "mode": "keyGen",
        "revision": "1.0",
        "testGroups": [
```

```
            {
                "tgId": 1,
                "curve": "ED-25519",
                "secretGenerationMode": "extra bits",
                "testType": "AFT",
                "tests": [
                    {
                        "tcId": 1
                    }
                ]
            }
        ]
    }
]
```

**Figure 7**

### 5.3. EDDSA 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 JSON elements of the Test Group JSON object.

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

**Table 11 — EDDSA keyVer Test Group JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | The test group identifier | integer |
| curve | The curve type used for the test vectors | string |
| testType | The testType for the group | string |
| tests | Array of individual test vector JSON objects, which are defined in Section 5.4 | array |

### 5.4. EDDSA 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 EDDSA test vector.

**Table 12 — Test Case JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tcId | Numeric identifier for the test case, unique across the entire vector set | integer |

| JSON Value | Description | JSON type |
|---|---|---|
| q | The encoded public key curve point | hex |

The following is an example of a prompt for EDDSA / keyVer / 1.0

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "EDDSA",
        "mode": "keyVer",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "curve": "ED-25519",
                "testType": "AFT",
                "tests": [
                    {
                        "tcId": 1,
                        "q":
 "227093C50F7D04A41121CEFDF076CC8B21D44E7506F341F8BFAB269CE06F2B7E",
                    }
                ]
            }
        ]
    }
]
```

**Figure 8**

## 5.5. EDDSA 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 JSON elements of the Test Group JSON object.

The test group for EDDSA / sigGen / 1.0 is as follows:

**Table 13 — EDDSA sigGen Test Group JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | The test group identifier | integer |
| curve | The curve type used for the test vectors | string |

| JSON Value | Description | JSON type |
|---|---|---|
| prehash | Whether or not Prehash EdDSA/HashEdDSA (vs normal/'pure' EdDSA) should be used for the test vectors | boolean |
| testType | The testType for the group | string |
| tests | Array of individual test vector JSON objects, which are defined in Section 5.6 | array |

### 5.6. EDDSA 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 EDDSA test vector.

**Table 14 — 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 the signature | hex |
| context | The context string defined in FIPS 186-5 sections 7.6 and 7.8 | hex |
| NOTE – The 'context' property will only be present for 1) normal/'pure' EdDSA signature generation tests that use Ed448 and 2) Prehash EdDSA/HashEdDSA signature generation tests that use Ed448 or Ed25519. | | |

The following is an example of a prompt for EDDSA / sigGen / 1.0

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "EDDSA",
        "mode": "sigGen",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "testType": "AFT",
                "curve": "ED-25519",
                "preHash": true,
                "tests": [
                    {
                        "tcId": 1,
```

```
                        "message": "A81C8A22735A260...",
                        "context": "12CBEF869A08BCD..."
                    }
                ]
            },
            {
                "tgId": 5,
                "testType": "BFT",
                "curve": "ED-25519",
                "preHash": false,
                "tests": [
                    {
                        "tcId": 41,
                        "message": "F27E9F9D"
                    },
                    {
                        "tcId": 42,
                        "message": "F27E9F9C"
                    },
                    {
                        "tcId": 43,
                        "message": "F27E9F9F"
                    }
                ]
            }
        ]
    }
]
```


**Figure 9**

## 5.7. EDDSA 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 JSON elements of the Test Group JSON object.

The test group for EDDSA / sigVer / 1.0 is as follows:

**Table 15 — EDDSA sigVer Test Group JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | The test group identifier | integer |
| curve | The curve type used for the test vectors | string |

| JSON Value | Description | JSON type |
|---|---|---|
| prehash | Whether or not Prehash EdDSA/HashEdDSA (vs normal/'pure' EdDSA) should be used for the test vectors | boolean |
| testType | The testType for the group | string |
| tests | Array of individual test vector JSON objects, which are defined in Section 5.8 | array |

### 5.8. EDDSA 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 EDDSA test vector.

**Table 16 — 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 the signature | hex |
| q | The encoded public key | hex |
| signature | The signature to verify | hex |

The following is an example of a prompt for EDDSA / sigVer / 1.0

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 0,
        "algorithm": "EDDSA",
        "mode": "sigVer",
        "revision": "1.0",
        "isSample": true,
        "testGroups": [
            {
                "tgId": 1,
                "testType": "AFT",
                "curve": "ED-25519",
                "preHash": false,
                "tests": [
                    {
                        "tcId": 1,
                        "message": "61524B41E89736DEE...",
                        "q": "14FB8D71A6CEDFC7B33109F...",
```

```
                        "signature": "283877CFDAFE61A..."
                    },
                    {
                        "tcId": 2,
                        "message": "43529BD72351015CA...",
                        "q": "99E318DCAD59F37DD3355EE...",
                        "signature": "F21BCB4898B32B6..."
                    }
                ]
            }
        ]
    }
]
```


**Figure 10**

# 6. 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 17 — 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 are defined in Section 6.1, Section 6.2, Section 6.3 and Section 6.4 | array |

## 6.1. EDDSA keyGen Test Group Responses

The following table describes the JSON object that represents a test group response for EDDSA / keyGen / 1.0.

**Table 18 — EDDSA keyGen Test Group Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | Unique numeric identifier for the test group | integer |
| tests | Array of JSON objects that represent each result, as defined by the table below | array |

The following table describes the JSON object that represents a test case response for EDDSA / keyGen / 1.0.

**Table 19 — EDDSA keyGen Test Case Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tcId | The test case identifier | integer |
| d | The encoded private key point | hex |
| q | The encoded public key point | hex |

The following is an example of the response for EDDSA / keyGen / 1.0

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

```
                "tgId": 1,
                "tests": [
                    {
                        "tcId": 1,
                        "q": "D51FB3D405A63622783...",
                        "d": "147BA261D11CD323331..."
                    }
                ]
            }
        ]
    }
]
```


**Figure 11**

## 6.2. EDDSA keyVer Test Group Responses

The following table describes the JSON object that represents a test group response for EDDSA / keyVer / 1.0.

**Table 20 — EDDSA keyVer Test Group Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | Unique numeric identifier for the test group | integer |
| tests | Array of JSON objects that represent each result, as defined by the table below | array |

The following table describes the JSON object that represents a test case response for EDDSA / keyVer / 1.0.

**Table 21 — EDDSA keyVer Test Case Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tcId | The test case identifier | integer |
| testPassed | Whether or not the key provided was valid | boolean |

The following is an example of the response for EDDSA / keyVer / 1.0


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

```
                "tests": [
                    {
                        "tcId": 1,
                        "testPassed": true
                    }
                ]
            }
        ]
    }
]
```

**Figure 12**

## 6.3. EDDSA sigGen Test Group Responses

The following table describes the JSON object that represents a test group response for EDDSA / sigGen / 1.0.

**Table 22 — EDDSA sigGen Test Group Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | Unique numeric identifier for the test group | integer |
| q | The encoded public key point used for signatures in the group | hex |
| tests | Array of JSON objects that represent each result, as defined by the table below | array |

The following table describes the JSON object that represents a test case response for EDDSA / sigGen / 1.0.

**Table 23 — EDDSA sigGen Test Case Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tcId | The test case identifier | integer |
| signature | The computed signature | hex |

The following is an example of the response for EDDSA / sigGen / 1.0

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "testGroups": [
            {
                "tgId": 1,
                "q": "4BA34FE699DBDC89750FF006...",
```

```
                "tests": [
                    {
                        "tcId": 1,
                        "signature": "772990B0..."
                    }
                ]
            },
            {
                "tgId": 5,
                "q": "ADD51513B67540E3A392721...",
                "tests": [
                    {
                        "tcId": 41,
                        "signature": "6EA857E..."
                    },
                    {
                        "tcId": 42,
                        "signature": "883B033..."
                    },
                    {
                        "tcId": 43,
                        "signature": "E402705..."
                    }
                ]
            }
        ]
    }
]
```


**Figure 13**

## 6.4. EDDSA sigVer Test Group Responses

The following table describes the JSON object that represents a test group response for EDDSA / sigVer / 1.0.

**Table 24 — EDDSA sigVer Test Group Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tgId | Unique numeric identifier for the test group | integer |
| tests | Array of JSON objects that represent each result, as defined by the table below | array |

The following table describes the JSON object that represents a test case response for EDDSA / sigVer / 1.0.

**Table 25 — EDDSA sigVer Test Case Response JSON Object**

| JSON Value | Description | JSON type |
|---|---|---|
| tcId | The test case identifier | integer |
| testPassed | Whether or not the signature provided was valid | boolean |

The following is an example of the response for EDDSA / sigVer / 1.0

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

**Figure 14**

## 7. 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 | 2018-08-01 | 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 (October 2019) *Digital Signature Standard (DSS)* (Gaithersburg, MD), October 2019. FIPS 186-5 (Draft). https://doi.org/10.6028/NIST.FIPS.186-5-draft.

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.