ACVP ECDSA 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 PUB 186 ECDSA 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 ECDSA 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 ECDSA implementations using ACVP.

2. Supported ECDSA Algorithms

The following ECDSA algorithms **MAY** be advertised by the ACVP compliant cryptographic module. The list is in the form "algorithm / mode / revision".

- ECDSA / keyGen / 1.0
- ECDSA / keyVer / 1.0
- ECDSA / sigGen / 1.0
- ECDSA / sigVer / 1.0
- ECDSA / keyGen / FIPS186-5
- ECDSA / keyVer / FIPS186-5
- ECDSA / sigGen / FIPS186-5
- ECDSA / sigVer / FIPS186-5
- DetECDSA / sigGen / FIPS186-5

2.1. Supported Conformances for ECDSA Algorithms

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

- ECDSA / sigGen / 1.0
- ECDSA / sigVer / 1.0
- ECDSA / sigGen / FIPS186-5
- ECDSA / sigVer / FIPS186-5
- DetECDSA / sigGen / FIPS186-5

3. Test Types and Test Coverage

3.1. Test Types

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

- ECDSA / keyGen / * "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.
- ECDSA / keyVer / * "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.
- ECDSA / sigGen / * "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.
- ECDSA / sigVer / * "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.
- DetECDSA / sigGen / FIPS186-5 "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. The random value used in the signature is generated according to the Deterministic ECDSA algorithm in [FIPS 186-5 (Draft)].

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [FIPS 186-4], [FIPS 186-5 (Draft)], [SP 800-89], and [SP 800-106].

3.2.1. Requirements Covered

- FIPS 186-4 — Section 3. General Discussion. Domain parameter generation, key generation, signature generation, and signature validation are all within scope of ACVP server testing.
- FIPS 186-4 — Section 6. The Eliliptic Curve Digital Signature Algorithm (ECDSA). The ACVP server **SHALL** allow testing with the recommended NIST curves. The ACVP server **SHALL** support a variety of curves/hash function for creation and delivery to/from the IUT. Key pair generation/verification testing **SHALL** be provided by the ACVP server. Both

Signature Generation and Validation testing mechanmisms **SHALL** be provided by the ACVP server.

- SP800-106 — Sections 3. Randomized Hashing and 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

- FIPS 186-4 — Section 3. General Discussion. Assurances of private key secrecy and ownership **SHALL NOT** be within scope of ACVP testing.
- FIPS 186-4 — Section 6. The Elliptic Curve Digital Signature Algorithm (ECDSA). 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 **SHALL NOT** provide testing for the validity of domain parameters as testing is (currently) limited to approved NIST curves. 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 ECDSA 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 ECDSA Validations

Each ECDSA implementation relies on other cryptographic primitives. For example, ECDSA sigGen 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 ECDSA Prerequisite Algorithms JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	a prerequisite algorithm	string	SHA, or DRBG
valValue	algorithm validation number	string	Actual number or “same”
prereqAlgVal	prerequisite algorithm validation	object with algorithm and valValue properties	See above

4.3. ECDSA Algorithm Registration Properties

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

Table 3 — ECDSA Algorithm Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	The ECDSA algorithm to be validated	string	See Section 2
mode	The ECDSA mdoe to be validated	string	See Section 2
revision	The algorithm testing revision to use	string	See Section 2
prereqVals	Prerequisite algorithm validations	array of prereqAlgVal objects	See Section 4.2
capabilities	The individual ECDSA Mode and Revision capabilities	array of capability objects	See Section 4.3.1, Section 4.3.2, Section 4.3.3, Section 4.3.4, Section 4.3.5
conformances	Used to denote the conformances that can	array of strings	See Section 2.1

JSON Value	Description	JSON type	Valid Values
	apply to specific modes of ECDSA		
componentTest	If the hash is performed outside of the boundary of the algorithm, setting this to true will cause the messages to be pre-hashed when provided by the server	boolean	true or false
NOTE – The 'componentTest' property is only valid for detECDSA / sigGen / FIPS186-5, ECDSA / sigGen / * and ECDSA / sigVer / * registrations.			

4.3.1. The keyGen Mode Capabilities

The ECDSA keyGen 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 ECDSA keyGen mode capability set is advertised as a self-contained JSON object.

4.3.1.1. keyGen Capabilities For Revision "1.0"

The complete list of ECDSA / keyGen / 1.0 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 4 — ECDSA keyGen 1.0 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 {"P-224", "P-256", "P-384", "P-521", "B-233", "B-283", "B-409", "B-571", "K-233", "K-283", "K-409", "K-571"}
secretGenerationMode	The method used to generate the randomness incoporated in the key.	array	Any non-empty subset of {"extra bits", "testing candidates"}
NOTE – The 'secretGenerationMode' can be found in [FIPS 186-4] Appendix B.4.			

Below is an example of the registration for ECDSA / keyGen / 1.0

```
{
    "algorithm": "ECDSA",
    "mode": "keyGen",
    "revision": "1.0",
    "prereqVals": [
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "curve": [
        "P-224",
        "P-256",
        "P-384",
        "P-521",
        "B-233",
        "B-283",
        "B-409",
        "B-571",
        "K-233",
        "K-283",
        "K-409",
        "K-571"
    ],
    "secretGenerationMode": [
        "extra bits",
        "testing candidates"
    ]
}
```

Figure 2

4.3.1.2. keyGen Capabilities For Revision "FIPS186-5"

The complete list of ECDSA / keyGen / FIPS186-5 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 5 — ECDSA keyGen FIPS186-5 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 {"P-224", "P-256", "P-384", "P-521"}
secretGenerationMode	The method used to generate the randomness	array	Any non-empty subset of {"extra bits", "testing candidates"}

JSON Value	Description	JSON type	Valid Values
	incoporated in the key.		
NOTE – The 'secretGenerationMode' can be found in [FIPS 186-5 (Draft)] Appendix A.2.			

Below is an example of the registration for ECDSA / keyGen / FIPS186-5

```
{
    "algorithm": "ECDSA",
    "mode": "keyGen",
    "revision": "FIPS186-5",
    "prereqVals": [
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "curve": [
        "P-224",
        "P-256",
        "P-384",
        "P-521"
    ],
    "secretGenerationMode": [
        "extra bits",
        "testing candidates"
    ]
}
```

Figure 3

4.3.2. The keyVer Mode Capabilities

The ECDSA keyVer 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 ECDSA keyVer mode capability set is advertised as a self-contained JSON object.

4.3.2.1. keyVer Capabilities For Revision "1.0"

The complete list of ECDSA / keyVer / 1.0 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 6 — ECDSA keyVer 1.0 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 {"P-192", "P-224", "P-256", "P-384", "P-521", "B-163", "B-233", "B-283", "B-409", "B-571", "K-163", "K-233", "K-283", "K-409", "K-571"}

Below is an example of the registration for ECDSA / keyVer / 1.0

```
{
    "algorithm": "ECDSA",
    "mode": "keyVer",
    "revision": "1.0",
    "prereqVals": [
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "curve": [
        "P-192",
        "P-224",
        "P-256",
        "P-384",
        "P-521",
        "B-163",
        "B-233",
        "B-283",
        "B-409",
        "B-571",
        "K-163",
        "K-233",
        "K-283",
        "K-409",
        "K-571"
    ]
}
```

Figure 4

4.3.2.2. keyVer Capabilities For Revision “FIPS186-5”

The complete list of ECDSA / keyVer / FIPS186-5 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 7 — ECDSA keyVer FIPS186-5 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 {”P-224”, “P-256”, “P-384”, “P-521”}

Below is an example of the registration for ECDSA / keyVer / FIPS186-5

```
{
    "algorithm": "ECDSA",
    "mode": "keyVer",
    "revision": "FIPS186-5",
    "prereqVals": [
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "curve": [
        "P-224",
        "P-256",
        "P-384",
        "P-521"
    ]
}
```

Figure 5

4.3.3. The sigGen Mode Capabilities

The ECDSA sigGen 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 ECDSA sigGen mode capability set is advertised as a self-contained JSON object.

4.3.3.1. sigGen Capabilities For Revision “1.0”

The complete list of ECDSA / sigGen / 1.0 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 8 — ECDSA sigGen 1.0 Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
curve	The curves supported with a particular set of hash algorithms.	array	Any non-empty subset of {"P-224", "P-256", "P-384", "P-521", "B-233", "B-283", "B-409", "B-571", "K-233", "K-283", "K-409", "K-571"}
hashAlg	The hash functions supported when signing a message for a particular set of curves.	array	Any non-empty subset of {"SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256", "SHA3-224", "SHA3-256", "SHA3-384", "SHA3-512"}
NOTE – Separate capability JSON objects in the array **MAY** represent different groupings of curves and hash algorithms. For example if one object in the 'capabilities' array has 'curve' "P-224" and 'hashAlg' "SHA2-224" while the next object has 'curve' "P-256" and 'hashAlg' "SHA2-256", then the tests generated will maintain those relations. A test group will be generated with cases for signatures over "P-224" using "SHA2-224" and a test group will be generated with cases for signatures over "P-256" using "SHA2-256".			

Below is an example of the registration for ECDSA / sigGen / 1.0

```
{
    "algorithm": "ECDSA",
    "mode": "sigGen",
    "revision": "1.0",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "componentTest": false,
    "capabilities": [
    {
        "curve": [
            "P-224",
            "P-256"
        ],
        "hashAlg": [
```

```
            "SHA2-224",
            "SHA2-256"
        ]
    },
    {
        "curve": [
            "P-512"
        ],
        "hashAlg": [
            "SHA3-512"
        ]
    }],
    "conformances": [
        "SP800-106"
    ]
}
```

Figure 6

4.3.3.2. sigGen Capabilities For Revision "FIPS186-5"

The complete list of ECDSA / sigGen / FIPS186-5 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 9 — ECDSA sigGen FIPS186-5 Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
curve	The curves supported with a particular set of hash algorithms.	array	Any non-empty subset of {"P-224", "P-256", "P-384", "P-521"}
hashAlg	The hash functions supported when signing a message for a particular set of curves.	array	Any non-empty subset of {"SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256", "SHA3-224", "SHA3-256", "SHA3-384", "SHA3-512", "SHAKE-128", "SHAKE-256"}
NOTE – Separate capability JSON objects in the array **MAY** represent different groupings of curves and hash algorithms. For example if one object in the 'capabilities' array has 'curve' "P-224" and 'hashAlg' "SHA2-224" while the next object has 'curve' "P-256" and 'hashAlg' "SHA2-256", then the tests generated will maintain those relations. A test group will be generated with cases for signatures over "P-224" using "SHA2-224" and a test group will be generated with cases for signatures over "P-256" using "SHA2-256".			

Below is an example of the registration for ECDSA / sigGen / FIPS186-5

```
{
    "algorithm": "ECDSA",
    "mode": "sigGen",
    "revision": "FIPS186-5",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "componentTest": false,
    "capabilities": [
    {
        "curve": [
            "P-224",
            "P-256"
        ],
        "hashAlg": [
            "SHA2-224",
            "SHA2-256"
        ]
    },
    {
        "curve": [
            "P-512"
        ],
        "hashAlg": [
            "SHA3-512"
        ]
    }],
    "conformances": [
        "SP800-106"
    ]
}
```

Figure 7

4.3.4. The sigVer Mode Capabilities

The ECDSA sigVer 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 ECDSA sigVer mode capability set is advertised as a self-contained JSON object.

4.3.4.1. sigVer Capabilities For Revision "1.0"

The complete list of ECDSA / sigVer / 1.0 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 10 — ECDSA sigVer Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
curve	The curves supported with a particular set of hash algorithms.	array	Any non-empty subset of {"P-192", "P-224", "P-256", "P-384", "P-521", "B-163", "B-233", "B-283", "B-409", "B-571", "K-163", "K-233", "K-283", "K-409", "K-571"}
hashAlg	The hash functions supported when signing a message for a particular set of curves.	array	Any non-empty subset of {"SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256", "SHA3-224", "SHA3-256", "SHA3-384", "SHA3-512"}
NOTE – Separate capability JSON objects in the array **MAY** represent different groupings of curves and hash algorithms. For example if one object in the 'capabilities' array has 'curve' "P-224" and 'hashAlg' "SHA2-224" while the next object has 'curve' "P-256" and 'hashAlg' "SHA2-256", then the tests generated will maintain those relations. A test group will be generated with cases for signatures over "P-224" using "SHA2-224" and a test group will be generated with cases for signatures over "P-256" using "SHA2-256".			

Below is an example of the registration for ECDSA / sigVer / 1.0

```
{
    "algorithm": "ECDSA",
    "mode": "sigVer",
    "revision": "1.0",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
```

```
    ],
    "componentTest": false,
    "capabilities": [
    {
        "curve": [
            "P-224",
            "P-256"
        ],
        "hashAlg": [
            "SHA2-224",
            "SHA2-256"
        ]
    },
    {
        "curve": [
            "P-512"
        ],
        "hashAlg": [
            "SHA3-512"
        ]
    }],
    "conformances": [
        "SP800-106"
    ]
}
```


Figure 8

4.3.4.2. sigVer Capabilities For Revision "FIPS186-5"

The complete list of ECDSA / sigVer / FIPS186-5 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 11 — ECDSA sigVer Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
curve	The curves supported with a particular set of hash algorithms.	array	Any non-empty subset of {"P-224", "P-256", "P-384", "P-521"}
hashAlg	The hash functions supported when signing a message for a particular set of curves.	array	Any non-empty subset of {"SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256", "SHA3-224", "SHA3-256", "SHA3-384", "SHA3-

JSON Value	Description	JSON type	Valid Values
			512", "SHAKE-128", "SHAKE-256"}

NOTE – Separate capability JSON objects in the array **MAY** represent different groupings of curves and hash algorithms. For example if one object in the 'capabilities' array has 'curve' "P-224" and 'hashAlg' "SHA2-224" while the next object has 'curve' "P-256" and 'hashAlg' "SHA2-256", then the tests generated will maintain those relations. A test group will be generated with cases for signatures over "P-224" using "SHA2-224" and a test group will be generated with cases for signatures over "P-256" using "SHA2-256".

Below is an example of the registration for ECDSA / sigVer / FIPS186-5

```
{
    "algorithm": "ECDSA",
    "mode": "sigVer",
    "revision": "FIPS186-5",
    "prereqVals": [{
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "componentTest": false,
    "capabilities": [
    {
        "curve": [
            "P-224",
            "P-256"
        ],
        "hashAlg": [
            "SHA2-224",
            "SHA2-256"
        ]
    },
    {
        "curve": [
            "P-512"
        ],
        "hashAlg": [
            "SHA3-512"
        ]
    }],
    "conformances": [
```

```
        "SP800-106"
    ]
}
```

Figure 9

4.3.5. The Deterministic ECDSA sigGen Mode Capabilities

The ECDSA sigGen 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 ECDSA sigGen mode capability set is advertised as a self-contained JSON object.

4.3.5.1. Deterministic ECDSA sigGen Capabilities For Revision "FIPS186-5"

The complete list of DetECDSA / sigGen / FIPS186-5 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 12 — Deterministic ECDSA sigGen FIPS186-5 Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
curve	The curves supported with a particular set of hash algorithms.	array	Any non-empty subset of {"P-224", "P-256", "P-384", "P-521"}
hashAlg	The hash functions supported when signing a message for a particular set of curves.	array	Any non-empty subset of {"SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256", "SHA3-224", "SHA3-256", "SHA3-384", "SHA3-512", "SHAKE-128", "SHAKE-256"}
NOTE – Separate capability JSON objects in the array **MAY** represent different groupings of curves and hash algorithms. For example if one object in the 'capabilities' array has 'curve' "P-224" and 'hashAlg' "SHA2-224" while the next object has 'curve' "P-256" and 'hashAlg' "SHA2-256", then the tests generated will maintain those relations. A test group will be generated with cases for signatures over "P-224" using "SHA2-224" and a test group will be generated with cases for signatures over "P-256" using "SHA2-256".			

Below is an example of the registration for detECDSA / sigGen / FIPS186-5

```
{
    "algorithm": "DetECDSA",
    "mode": "sigGen",
    "revision": "FIPS186-5",
    "prereqVals": [{
```

```
            "algorithm": "SHA",
            "valValue": "123456"
        },
        {
            "algorithm": "DRBG",
            "valValue": "123456"
        }
    ],
    "componentTest": false,
    "capabilities": [
    {
        "curve": [
            "P-224",
            "P-256"
        ],
        "hashAlg": [
            "SHA2-224",
            "SHA2-256"
        ]
    },
    {
        "curve": [
            "P-512"
        ],
        "hashAlg": [
            "SHA3-512"
        ]
    }],
    "conformances": [
        "SP800-106"
    ]
}
```


Figure 10

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 PUB 186 ECDSA 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 13 — 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 11

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 ECDSA / sigGen / 1.0, ECDSA / keyVer / FIPS186-5, etc. This section describes the JSON schema for a test vector set used with ECDSA 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 14 — ECDSA 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. ECDSA keyGen Test Vectors

6.1.1. ECDSA 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 ECDSA / keyGen / * is as follows:

Table 15 — ECDSA keyGen Test Group JSON Object

JSON Value	Description	JSON type
testType	The test operation performed	string
curve	The curve type used for the test group	string

JSON Value	Description	JSON type
secretGenerationMode	The secret generation mode used for the group	string
tests	Array of individual test vector JSON objects, which are defined in Section 6.1.2	array

6.1.2. ECDSA keyGen Test Groups 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 ECDSA test vector.

Table 16 — ECDSA keyGen Test Case JSON Object

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

The following is an example JSON object sent from the server to the client for ECDSA / keyGen. While the example will specify a revision, the format is identical for both revisions available.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "ECDSA",
        "mode": "keyGen",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "curve": "P-224",
                "secretGenerationMode": "extra bits",
                "tests": [
                    {
                        "tcId": 1
                    }
                ]
            }
        ]
    }
```

```
]
```

Figure 12

6.2. ECDSA keyVer Test Vectors

6.2.1. ECDSA 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 ECDSA / keyVer / * is as follows:

Table 17 — ECDSA keyVer Test Group JSON Object

JSON Value	Description	JSON type
testType	The test operation performed	string
curve	The curve type used for the test group	string
tests	Array of individual test vector JSON objects, which are defined in Section 6.2.2	array

6.2.2. ECDSA keyVer Test Groups 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 ECDSA test vector.

Table 18 — ECDSA keyVer Test Case JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
qx	The public key curve point x	hex
qy	The public key curve point y	hex

The following is an example JSON object sent from the server to the client for ECDSA / keyVer. While the example will specify a revision, the format is identical for both revisions available.

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

```
        "mode": "keyVer",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "curve": "P-192",
                "tests": [
                    {
                        "tcId": 1,
                        "qx":
 "01ED77E3F1591D2EC730D0ED6D592F8DD24158D0E696408DBD",
                        "qy":
 "BF31C6463EB1B6B55C8930550B88CF8D1F6432A832B40FB4"
                    }
                ]
            }
        ]
    }
]
```


Figure 13

6.3. ECDSA sigGen Test Vectors

6.3.1. ECDSA 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 ECDSA / sigGen / * is as follows:

Table 19 — ECDSA sigGen Test Group JSON Object

JSON Value	Description	JSON type
testType	The test operation performed	string
curve	The curve type used for the test vectors	string
hashAlg	SHA version used	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 6.3.2	array

6.3.2. ECDSA sigGen Test Groups 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 ECDSA test vector.

Table 20 — ECDSA 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
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 'randomValue' and 'randomValueLen' will only be present if the 'conformance' "SP800-106" is present in the group.		

The following is an example JSON object sent from the server to the client for ECDSA / sigGen. While the example will specify a revision, the format is identical for both revisions available.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "ECDSA",
        "mode": "sigGen",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "curve": "P-224",
                "hashAlg": "SHA2-224",
                "tests": [
                    {
                        "tcId": 1,
                        "message": "AB6F57713A3BD323B4AFDCFBE202EE0..."
                    }
                ]
            },
            {
```

```
                "tgId": 2,
                "curve": "P-224",
                "hashAlg": "SHA2-224",
                "conformance": "SP800-106",
                "tests": [
                    {
                        "tcId": 2,
                        "message": "23B4AFDCFBE202EE00A9CF5C787D19FD90..."
                    }
                ]
            }
        ]
    }
]
```


Figure 14

6.4. ECDSA sigVer TestVectors

6.4.1. ECDSA 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 ECDSA / sigVer / * is as follows:

Table 21 — ECDSA sigVer Test Group JSON Object

JSON Value	Description	JSON type
testType	The test operation performed	string
curve	The curve type used for the test vectors	string
hashAlg	SHA version used	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 6.4.2	array

6.4.2. ECDSA sigVer Test Groups 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 ECDSA test vector.

Table 22 — ECDSA 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
qx	The public key curve point x	hex
qy	The public key curve point y	hex
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 'randomValue' and 'randomValueLen' will only be present if the 'conformance' "SP800-106" is present in the group.		

The following is an example JSON object sent from the server to the client for ECDSA / sigVer. While the example will specify a revision, the format is identical for both revisions available.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "ECDSA",
        "mode": "sigVer",
        "revision": "1.0",
        "testGroups": [
            {
                "tgId": 1,
                "curve": "P-192",
                "hashAlg": "SHA-1",
                "tests": [
                    {
                        "tcId": 1,
                        "message": "D38A81D0C5201BA4A06A8C4760AC15DB266B1...",
                        "qx": "B08AFEAC74E42C66EBAF13807E2EB5769F5123645C...",
                        "qy": "55847857E5E48025BE9053952E0E1ECFB1D883CF9F...",
                        "r": "E31121E544D476DC3FA79B4DCB0A7252B6E80468BBF...",
                        "s": "6E3F47F2327E36AD936E0F4BE245C05F264BA9300E9..."
                    }
```

```
                ]
            },
            {
                "tgId": 2,
                "curve": "P-192",
                "hashAlg": "SHA-1",
                "conformance": "SP800-106",
                "tests": [
                    {
                        "tcId": 2,
                        "message": "D38A81D04A06A8C4760AC15DB266B17B48B...",
                        "randomValue": "1527E0FE37FD1162F5DD0D975E83C0D...",
                        "randomValueLen": 1024
                        "qx": "D1E896486D9D986A464D3469941F93FC65556E2CB...",
                        "qy": "ADCB8D50375DC76907195B6AF6C06F...",
                        "r": "6D9D986A464D3469941F93FC65556E2CB8AB5F113...",
                        "s": "8E713EB6106EF0E19E241DB4B4831E06437E5C..."
                    }
                ]
            }
        ]
    }
]
```


Figure 15

6.5. Deterministic ECDSA sigGen Test Vectors

6.5.1. Deterministic ECDSA 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 DetECDSA / sigGen / * is as follows:

Table 23 — Deterministic ECDSA sigGen Test Group JSON Object

JSON Value	Description	JSON type
testType	The test operation performed	string
curve	The curve type used for the test vectors	string
hashAlg	SHA version used	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 6.3.2	array

6.5.2. detECDSA 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 ECDSA test vector.

Table 24 — Deterministic ECDSA 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
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 'randomValue' and 'randomValueLen' will only be present if the 'conformance' "SP800-106" is present in the group.		

The following is an example JSON object sent from the server to the client for DetECDSA / sigGen / FIPS186-5.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "algorithm": "DetECDSA",
        "mode": "sigGen",
        "revision": "FIPS186-5",
        "testGroups": [
            {
                "tgId": 1,
                "curve": "P-224",
                "hashAlg": "SHA2-224",
                "tests": [
                    {
                        "tcId": 1,
```

```
                        "message": "AB6F57713A3BD323B4AFDCFBE202EE0..."
                    }
                ]
            },
            {
                "tgId": 2,
                "curve": "P-224",
                "hashAlg": "SHA2-224",
                "conformance": "SP800-106",
                "tests": [
                    {
                        "tcId": 2,
                        "message": "23B4AFDCFBE202EE00A9CF5C787D19FD90..."
                    }
                ]
            }
        ]
    }
]
```


Figure 16

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 25 — 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 26	array

An example of this is the following

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

Figure 17

The 'testGroups' section is used to organize the ACVP client response in a similar manner to how it distributes vectors.

Table 26 — 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 27, Table 28, Table 30, Table 31 or Table 33	array

An example of this is the following

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

Figure 18

7.1. ECDSA keyGen Test Vector Responses

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 ECDSA / keyGen / * test vector.

Table 27 — ECDSA keyGen Test Case Response JSON Object

JSON Value	Description	JSON type
tcId	The test case identifier	integer
d	The private key	hex
qx	The public key curve point x	hex
qy	The public key curve point y	hex

The following is an example JSON test vector response object for ECDSA / keyGen. While the example will not specify a revision, the format is identical for both revisions available.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "testGroups": [
            {
                "tgId": 1,
                "tests": [
                    {
                        "tcId": 1,
                        "qx": "7B1AA6BE712542282B8D088C23316...",
                        "qy": "BCC9213347A7F988A2FF9EF14C852...",
                        "d": "38524F26660BBA72E74EB39DEF3855..."
                    }
                ]
            }
        ]
    }
]
```

Figure 19

7.2. ECDSA keyVer Test Vector Responses

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 ECDSA / keyVer / * test vector.

Table 28 — ECDSA 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 ECDSA / keyVer. While the example will not specify a revision, the format is identical for both revisions available.

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

Figure 20

7.3. ECDSA sigGen Test Vector Responses

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

Table 29 — ECDSA sigGen Test Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
qx	The x component of the public key	hex
qy	The y component of the public key	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 ECDSA / sigGen / * test vector.

Table 30 — ECDSA 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' **SHALL** only be present in test groups where the corresponding test group in the prompt had the 'conformance' property set to "SP800-106".		

The following is an example JSON test vector response object for ECDSA / sigGen. While the example will not specify a revision, the format is identical for both revisions available.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "testGroups": [
            {
                "tgId": 1,
                "qx": "3B1D9E4D986F651C3C213B2A1304693BDB...",
                "qy": "E56F7B7C9E6355E573B7B3B6C0E1ECD70E...",
                "tests": [
                    {
                        "tcId": 1,
                        "r": "3E2A9588DF3D3F11B16368A30C8...",
                        "s": "C6E4A8C51E0A0E11C4C6D6F8F3C..."
                    }
                ]
            },
            {
                "tgId": 2,
                "qx": "A1304693BDBA632CB93A3B8BA632CB93A3...",
                "qy": "ECD70E4ABBA632CB93A3BA632CB93A3DF1...",
                "tests": [
                    {
                        "tcId": 2,
                        "r": "3E2A9588DF3D3F11B16368A30C8...",
                        "s": "C6E4A8C51E0A0E11C4C6D6F8F3C...",
```

```
                        "randomValue": "0A0E11C4C6D6F8F3C..."
                        "randomValueLen": 1024
                    }
                ]
            }
        ]
    }
]
```

Figure 21

7.4. ECDSA sigVer Test Vector Responses

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 ECDSA / sigVer / * test vector.

Table 31 — ECDSA sigVer Test Case Response JSON Object

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

The following is an example JSON test vector response object for ECDSA / sigVer. While the example will not specify a revision, the format is identical for both revisions available.

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

Figure 22

7.5. Deterministic ECDSA sigGen Test Vector Responses

The test groups for detECDSA / sigGen / FIPS186-5 contain public key properties. The groups can be described using the following table.

Table 32 — Deterministic ECDSA sigGen Test Group Response JSON Object

JSON Value	Description	JSON type
tgId	The test group identifier	integer
qx	The x component of the public key	hex
qy	The y component of the public key	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 DetECDSA / sigGen / FIPS186-5 test vector.

Table 33 — Deterministic ECDSA 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' **SHALL** only be present in test groups where the corresponding test group in the prompt had the 'conformance' property set to "SP800-106".		

The following is an example JSON test vector response object for DetECDSA / sigGen / FIPS186-5.

```
[
    {
        "acvVersion": <acvp-version>
    },
    {
        "vsId": 1564,
        "testGroups": [
            {
                "tgId": 1,
                "qx": "3B1D9E4D986F651C3C213B2A1304693BDB...",
                "qy": "E56F7B7C9E6355E573B7B3B6C0E1ECD70E...",
                "tests": [
```

```
                    {
                        "tcId": 1,
                        "r": "3E2A9588DF3D3F11B16368A30C8...",
                        "s": "C6E4A8C51E0A0E11C4C6D6F8F3C..."
                    }
                ]
            },
            {
                "tgId": 2,
                "qx": "A1304693BDBA632CB93A3B8BA632CB93A3...",
                "qy": "ECD70E4ABBA632CB93A3BA632CB93A3DF1...",
                "tests": [
                    {
                        "tcId": 2,
                        "r": "3E2A9588DF3D3F11B16368A30C8...",
                        "s": "C6E4A8C51E0A0E11C4C6D6F8F3C...",
                        "randomValue": "0A0E11C4C6D6F8F3C..."
                        "randomValueLen": 1024
                    }
                ]
            }
        ]
    }
]
```


Figure 23

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	2016-06-01	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 (July 2013) *Digital Signature Standard (DSS)* (Gaithersburg, MD), July 2013. FIPS 186-4. https://doi.org/10.6028/NIST.FIPS.186-4.

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.