ACVP RSA Algorithm JSON Specification

Christopher Celi
Information Technology Laboratory
Computer Security Division

November 01, 2019

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

Abstract

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

2. Supported RSA Modes

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

Table 1 — Supported RSA Algorithm Modes JSON Values

Algorithm	Mode	Revision	Standard
“RSA”	“keyGen”	“FIPS186-4”	[FIPS 186-4]
“RSA”	“keyGen”	“FIPS186-5”	[FIPS 186-5 (Draft)]
“RSA”	“sigGen”	“FIPS186-4”	[FIPS 186-4]
“RSA”	“sigGen”	“FIPS186-5”	[FIPS 186-5 (Draft)]
“RSA”	“sigVer”	“FIPS186-5”	[FIPS 186-5 (Draft)]
“RSA”	“sigVer”	“FIPS186-4”	[FIPS 186-4]
“RSA”	“sigVer”	“FIPS186-2”	[FIPS 186-2]
“RSA”	“signaturePrimitive”	“1.0”	[FIPS 186-5 (Draft)]
“RSA”	“decryptionPrimitive”	“1.0”	[SP 800-56B]

These will be refered to as algorithm / mode / revision throughout the document, for example RSA / keyGen / 1.0.

2.1. Supported Conformances for RSA Algorithms

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

- RSA / sigGen / FIPS186-4
- RSA / sigGen / FIPS186-5
- RSA / sigVer / FIPS186-2
- RSA / sigVer / FIPS186-4
- RSA / sigVer / FIPS186-5

3. Test Types and Test Coverage

This section describes the design of the tests used to validate RSA implementations.

3.1. Test Types

There are multiple test types depending on the algorithm and mode. Each has a specific value to be used in the testType field. The testType field definitions are:

"AFT" — Algorithm Functional Test. These tests can be processed by the client using a normal 'generate_key', 'sign', 'verify' or 'decrypt' operation depending on the mode. AFTs cause the implementation under test to exercise normal operations. In all cases, random data is used. The functional tests are designed to verify that the logical components of the process are operating correctly.

"GDT" — Generated Data Test. These tests require the client to generate all inputs and outputs in order to demonstrate their full capabilities in cases where input from the server might not be applicable.

"KAT" — Known Answer Test. These tests are staticly defined and are included in every vector set that matches their capabilities.

3.2. Test Types Per Mode

As each mode requires different tests, how each test type is processed may change depending on the mode.

ACVP servers that support RSA keyGen **MAY** support AFTs, GDTs, and KATs. The AFTs and GDTs **REQUIRE** the client to generate RSA key pairs based on the test group properties provided. The KATs **REQUIRE** the client to attempt to generate a key with specific starting 'p' and 'q' values via the Random Probable Prime method defined in [FIPS 186-4] and [FIPS 186-5 (Draft)].

ACVP servers that support RSA sigGen **MAY** support GDTs. The GDTs **REQUIRE** the client to generate a key pair and sign a message provided by the server based on the test group properties provided. The client provides the signature and public key to the server for verification.

ACVP servers that support RSA sigVer **MAY** support GDTs. The GDTs **REQUIRE** the client to verify provided signatures given a message and public key. The client provides the verification result back to the server.

ACVP servers that support RSA LegacysigVer **MAY** support GDTs. The GDTs **REQUIRE** the client to verify provided signatures given a message and public key. The client provides the verification result back to the server.

ACVP servers that support RSA SignaturePrimitive **MAY** support AFTs. The AFTs **REQUIRE** the client to determine whether or not the provided message can be signed with the provided public key. If so, also provide the signature for verification.

ACVP servers that support RSA DecryptionPrimitive **MAY** support AFTs. The AFTs **REQUIRE** the client to determine whether or not the provided ciphertext can be decrypted with the provided public key. If so, also provide the plaintext for verification.

3.3. Test Coverage

The tests described in this document are intended to ensure an implementation conformant with the requirements listed below. For clarity, related requirements not covered are also provided.

3.3.1. Requirements Covered

- FIPS 186-4 Section 3 General Discussion. Key generation, signature generation, and signature validation are all within scope of ACVP server testing.
- FIPS 186-4 Section 5 The RSA Digital Signature Algorithm. The ACVP server provides a means of testing the generation of RSA keys. The ACVP server **SHALL** support a variety of RSA capabilities functionS for the creation and delivery of tests to/from the IUT. Key pair generation testing **SHALL** be provided by the ACVP server. Both Signature Generation and Validation testing mechanisms **SHALL** be provided by the ACVP server.
- FIPS 186-2. The ACVP server MAY provide a means of testing legacy RSA functions such as RSA sigVer. This testing is provided to ensure an IUT is capable of verifying a signature that is no longer approved for generation, given the same capabilities.
- SP800-106 Section 3 Randomized Hashing and Section 4 Digital Signatures Using Randomized Hashing. The IUT **SHALL** be provided or provide a random value that should be used to “randomize” a message prior to signing and/or verifying an original message.

3.3.2. 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 5 The RSA Digital Signature Algorithm. 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. 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 RSA 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 2 — 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 RSA Validations

Each RSA implementation relies on other cryptographic primitives. For example, RSA keyGen often 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 3 — Required RSA Prerequisite Algorithms JSON Values

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

4.3. RSA Algorithm Registration Properties

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

Table 4 — RSA Algorithm Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	The RSA algorithm to be validated	string	See Section 2
mode	The mode 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 RSA capabilities	array of capability objects	See Section 4.4, Section 4.6, Section 4.7, Section 4.9, or Section 4.8
infoGeneratedByServer	This flag indicates that the server is responsible for	boolean	true/false

JSON Value	Description	JSON type	Valid Values
	generating inputs for Key Generation tests		
pubExpMode	Supports fixed or random public key exponent e	string	“fixed” or “random”
fixedPubExp	The value of the public key exponent e in hex if pubExpMode is “fixed”	hex	
keyFormat	The preferred private key format. The “standard” format has “p”, “q”, and “d” as the components of the private key. The “crt” (Chinese Remainder Theorem) format has “p”, “q”, “dmp1” (d modulo p-1), “dmq1” (d modulo q-1), and “iqmp” (inverse q modulo p) as the components	string	“standard” or “crt”

NOTE 1 – The ‘infoGeneratedByServer’ property is only valid for RSA / keyGen / * registrations.

NOTE 2 – The ‘pubExpMode’ and if the property is set to “fixed”, the ‘fixedPubExp’, are only valid for RSA / keyGen / * , and RSA / sigVer / * registrations.

NOTE 3 – The ‘keyFormat’ property is only valid for RSA / keyGen / * registrations.

4.4. Property Registration RSA keyGen FIPS186-4

The RSA keyGen FIPS186-4 capabilities are advertised as JSON objects within the ‘capabilities’ property.

4.4.1. keyGen Registration Table

A registration for RSA / keyGen / FIPS186-4 **SHALL** use these properties

Table 5 — RSA keyGen FIPS186-4 Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
randPQ	Key Generation mode to be validated. Random P and Q primes generated as (see [FIPS 186-4]): provable primes (Appendix B.3.2); probable primes (Appendix B.3.3); provable primes with conditions (Appendix B.3.4); provable/probable primes with conditions (Appendix B.3.5); probable primes with conditions (Appendix B.3.6)	string	"B.3.2", "B.3.3", "B.3.4", "B.3.5", "B.3.6"
properties	An array of objects containing properties for all supported moduli, primality test, and hash algorithms for a single key generation mode	array	
modulo	supported RSA modulo for the randPQ mode — see [FIPS 186-4], Appendix B.3	integer	2048, 3072 or 4096
hashAlg	Supported hash algorithms for the randPQ mode — see [FIPS 186-4], Appendix B.3. Needed for "B.3.2", "B.3.4" and "B.3.5"	array	any non-empty subset of {"SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}
primeTest	Primality test rounds of Miller-Rabin from Table C.2 or Table C.3 in [FIPS 186-4], Appendix C.3	array	any non-empty subset of {"tblC2", "tblC3"}. Needed for "B.3.3", "B.3.5" and "B.3.6"

An example of this is the following

```
{
    "algorithm": "RSA",
    "mode": "keyGen",
    "revision": "FIPS186-4",
    "prereqVals": [{"algorithm": "DRBG", "valValue": "1234"}, {"algorithm":
 "SHA", "valValue": "5678"}],
    "infoGeneratedByServer": false,
    "pubExpMode": "random",
    "keyFormat": "crt",
    "capabilities": [
```

```
{
    "randPQ": "B.3.2",
    "properties": [
        {
            "modulo": 2048,
            "hashAlg": [
                "SHA2-224"
            ]
        }
    ]
},
{
    "randPQ": "B.3.4",
    "properties": [
        {
            "modulo": 3072,
            "hashAlg": [
                "SHA2-224"
            ]
        }
    ]
},
{
    "randPQ": "B.3.3",
    "properties": [
        {
            "modulo": 2048,
            "primeTest": [
                "tblC2"
            ]
        }
    ]
},
{
    "randPQ": "B.3.6",
    "properties": [
        {
            "modulo": 3072,
            "primeTest": [
                "tblC2",
                "tblC3"
            ]
        }
    ]
},
{
```

```
            "randPQ" : "B.3.5",
            "properties" : [
                {
                    "modulo" : 4096,
                    "hashAlg" : [
                        "SHA2-512"
                    ],
                    "primeTest": [
                        "tblC3"
                    ]
                }
            ]
        }
    ]
}
```

Figure 2

4.5. Property Registration RSA keyGen FIPS186-5

The RSA / keyGen / FIPS186-5 capabilities are advertised as JSON objects within the 'capabilities' property.

4.5.1. keyGen Registration Table

A registration for RSA / keyGen / FIPS186-5 **SHALL** use these properties

Table 6 — RSA keyGen FIPS186-5 Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
randPQ	Key Generation mode to be validated. Random P and Q primes generated as (see [FIPS 186-5 (Draft)]): provable primes; probable primes; provable primes with auxiliary provable primes; probable primes with auxiliary provable primes; probable primes with auxiliary probable primes	string	"provable", "probable", "provableWithProvableAux", "probableWithProvableAux", "probableWithProbableAux"

JSON Value	Description	JSON type	Valid Values
properties	An array of objects containing properties for all supported moduli, primality test, and hash algorithms for a single key generation mode	array	
modulo	supported RSA modulo for the randPQ mode — see [FIPS 186-5 (Draft)]	integer	2048, 3072, 4096 or 8192
hashAlg	Supported hash algorithms for the randPQ mode — see [FIPS 186-5 (Draft)]. Needed for any 'randPQ' with provable primes	array	any non-empty subset of {"SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}
primeTest	Primality test rounds of Miller-Rabin from [FIPS 186-5 (Draft)]. Needed for any 'randPQ' with probable primes	array	any non-empty subset of {"2pow100", "2powSecStr"}
pMod8	The result of the evaluation of the generated p prime, p % 8	integer	0, 1, 3, 5, 7
qMod8	The result of the evaluation of the generated q prime, q % 8	integer	0, 1, 3, 5, 7
NOTE – The properties 'pMod8' and 'qMod8' with a value of 0, means that no modulus check will be performed on the generated primes p and q.			

The following is an example

```
{
    "algorithm": "RSA",
    "mode": "keyGen",
```

```
    "revision": "FIPS186-5",
    "prereqVals": [{"algorithm": "DRBG", "valValue": "1234"}, {"algorithm":
 "SHA", "valValue": "5678"}],
    "infoGeneratedByServer": false,
    "pubExpMode": "random",
    "keyFormat": "crt"
    "capabilities": [
        {
            "randPQ": "provable",
            "properties": [
                {
                    "modulo": 2048,
                    "hashAlg": [
                        "SHA2-224"
                    ],
                    "pMod8": 1,
                    "qMod8": 1
                }
            ]
        },
        {
            "randPQ": "probable",
            "properties": [
                {
                    "modulo": 2048,
                    "primeTest": [
                        "2pow100"
                    ],
                    "pMod8": 0,
                    "qMod8": 3
                }
            ]
        }
    ]
}
```

Figure 3

4.6. RSA sigGen Mode Capabilities

The RSA / sigGen / * mode capabilities are advertised as JSON objects within the ‘capabilities’ array as part of the ‘capability_exchange’ element of the ACVP JSON registration message. See the ACVP specification for details on the registration message.

Each RSA sigGen mode capability is advertised as a self-contained JSON object consisting of the algorithm, mode, and capabilities array. The capabilities array may contain multiple elements, each pertaining to a sigType that is supported by the client for the RSA mode being advertised.

The following table defines the capabilities that may be advertised by the ACVP compliant crypto modules.

4.6.1. RSA sigGen FIPS186-4 Capabilities Table

The following RSA / sigGen / FIPS186-4 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 7 — Supported RSA sigGen FIPS186-4 JSON Values

JSON value	Description	JSON type	Valid values
sigType	supported RSA signature types — see [FIPS 186-4], Section 5	string	one of {"ansx9.31", "pkcs1v1.5", "pss"}
properties	RSA signature generation parameters — see [FIPS 186-4], Section 5	array	modulo, hashAlg, and saltLen (when sigType is "pss")
modulo	supported RSA modulo for signature generation — see [FIPS 186-4], Section 5	integer	any one of the supported modulo sizes {2048, 3072, 4096}
hashPair	supported hash algorithms and optional salt length for signature generation for this sigType and modulo# — see [SP 800-131A], Section 9	array	an array of objects containing a hashAlg and an optional saltLen
hashAlg	supported hash algorithms for this sigType and modulo — see [SP 800-131A], Section 9	array	any non-empty subset of {"SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}
saltLen	supported salt lengths for PSS signature generation — see [FIPS 186-4], Section 5.5	integer	See the note below
NOTE – The 'saltLen' property for each hash algorithm **SHALL** only be present if the 'sigType' is "pss". The values allowed in PSS signature generation is between 0 and the length of the corresponding hash function output block (in bytes), the end points included.			

The following is an example of a registration for RSA / sigGen / FIPS186-4

```
{
    "algorithm": "RSA",
    "mode": "sigGen",
    "revision": "FIPS186-4",
    "prereqVals": [{"algorithm": "DRBG", "valValue": "same"}, {"algorithm":
 "SHA", "valValue": "same"}],
    "conformances": [
        "SP800-106"
    ],
    "capabilities" :
    [
        {
            "sigType" : "ansx9.31",
            "properties" :
            [
                {
                    "modulo" : 2048,
                    "hashPair" : [
                        {
                            "hashAlg" : "SHA2-224"
                        }
                    ]
                },
                {
                    "modulo" : 3072,
                    "hashPair" : [
                        {
                            "hashAlg" : "SHA2-256"
                        },
                        {
                            "hashAlg" : "SHA2-512"
                        }
                    ]
                }
            ]
        },
        {
            "sigType" : "pkcs1v1.5",
            "properties" :
            [
                {
                    "modulo" : 4096,
                    "hashPair" : [
                        {
                            "hashAlg" : "SHA2-224"
                        },
```

```
                        {
                            "hashAlg" : "SHA2-256"
                        }
                    ]
                }
            ]
        },
        {
            "sigType" : "pss",
            "properties" :
            [
                {
                    "modulo" : 3072,
                    "hashPair" : [
                        {
                            "hashAlg" : "SHA2-224",
                            "saltLen" : 28
                        },
                        {
                            "hashAlg" : "SHA2-256",
                            "saltLen" : 32
                        },
                        {
                            "hashAlg" : "SHA2-512",
                            "saltLen" : 64
                        }
                    ]
                }
            ]
        }
    ]
}
```

Figure 4

4.6.2. RSA sigGen FIPS186-5 Capabilities Table

The following RSA / sigGen / FIPS186-5 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 8 — Supported RSA sigGen FIPS186-5 JSON Values

JSON value	Description	JSON type	Valid values
sigType	supported RSA signature types — see [FIPS 186-5 (Draft)], Section 5	string	one of {”pkcs1v1.5”, “pss”}

JSON value	Description	JSON type	Valid values
properties	RSA signature generation parameters — see [FIPS 186-5 (Draft)], Section 5	array	modulo, hashAlg, and saltLen (when sigType is “pss”)
modulo	supported RSA modulo for signature generation — see [FIPS 186-5 (Draft)], Section 5	integer	any one of the supported modulo sizes {2048, 3072, 4096}
maskFunction	the mask function used, only valid for PSS	array	any non-empty subset of {”mgf1”, “shake-128”, “shake-256”}
hashPair	supported hash algorithms and optional salt length for signature generation for this sigType and modulo# — see [SP 800-131A], Section 9	array	an array of objects containing a hashAlg and an optional saltLen
hashAlg	supported hash algorithms for this sigType and modulo# — see [SP 800-131A], Section 9	array	any non-empty subset of {”SHA2-224”, “SHA2-256”, “SHA2-384”, “SHA2-512”, “SHA2-512/224”, “SHA2-512/256”}
saltLen	supported salt lengths for PSS signature generation — see [FIPS 186-5 (Draft)], Section 5.4	integer	See the note below
NOTE – The ‘saltLen’ property for each hash algorithm **SHALL** only be present if the ‘sigType’ is “pss”. The values allowed in PSS signature generation is between 0 and the length of the corresponding hash function output block (in bytes), the end points included.			

For an example of the RSA / sigGen / FIPS186-5 registration see the following abbreviated example for PSS

```
{
    "algorithm": "RSA",
    "mode": "sigGen",
    "revision": "FIPS186-5",
    "prereqVals": [{"algorithm": "DRBG", "valValue": "same"}, {"algorithm":
 "SHA", "valValue": "same"}],
    "conformances": [
        "SP800-106"
    ],
    "capabilities" :
```

```
    [
        {
            "sigType" : "pss",
            "properties" :
            [
                {
                    "modulo" : 3072,
                    "maskFunction": [
                        "SHAKE-128", "MGF1"
                    ]
                    "hashPair" : [
                        {
                            "hashAlg" : "SHA2-224",
                            "saltLen" : 28
                        },
                        {
                            "hashAlg" : "SHA2-256",
                            "saltLen" : 32
                        },
                        {
                            "hashAlg" : "SHA2-512",
                            "saltLen" : 64
                        }
                    ]
                }
            ]
        }
    ]
}
```

Figure 5

4.7. RSA sigVer Mode Capabilities

The RSA / sigVer / * mode capabilities are advertised as JSON objects within the array of ‘capabilities’ as part of the ‘capability_exchange’ element of the ACVP JSON registration message. See the ACVP specification for details on the registration message.

Each RSA / sigVer / * mode capability is advertised as a self-contained JSON object consisting of the algorithm, mode, and capabilities array. The capabilities array may contain multiple elements, each pertaining to a sigType that is supported by the client for the RSA mode being advertised.

The following table defines the capabilities that may be advertised by the ACVP compliant crypto modules.

4.7.1. RSA sigVer FIPS186-2 Capabilities Table

The following RSA / sigVer / FIPS186-2 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 9 — Supported RSA sigVer FIPS186-2 Capabilities

JSON value	Description	JSON type	Valid values
sigType	supported RSA signature types — see [FIPS 186-2], Section 5	string	one of {"ansx9.31", "pkcs1v1.5", "pss"}
properties	RSA signature verification parameters — see [FIPS 186-2], Section 5	array	modulo, hashAlg, and saltLen (when sigType is "pss")
modulo	supported RSA modulo for signature verification	integer	any one of the supported modulo sizes {1024, 2048, 3072, 4096}
hashPair	supported hash algorithms and optional salt length for signature verification for this sigType and modulo# -#see [SP 800-131A], Section 9	array	an array of objects containing a hashAlg and an optional saltLen
hashAlg	supported hash algorithms for this sigType and modulo# -# see [SP 800-131A], Section 9	array	any non-empty subset of {"SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}
saltLen	supported salt lengths for PSS signature verification — see [FIPS 186-4], Section 5.5	integer	See the note below
NOTE – The 'saltLen' property for each hash algorithm **SHALL** only be present of the 'sigType' is "pss". The values allowed in PSS signature generation is between 0 and the length of the corresponding hash function output block (in bytes), the end points included except for when 1024 is the modulo size. In this case, the maximum 'saltLen' for SHA2-512 is 62 bytes.			

For an example of the registration for RSA / sigVer / FIPS186-2, see the following example for RSA / sigVer / FIPS186-4. The formats are identical even though the individual allowed values might change.

4.7.2. RSA sigVer FIPS186-4 Capabilities Table

The following RSA / sigVer / FIPS186-4 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 10 — Supported RSA sigVer FIPS186-4 Capabilities

JSON value	Description	JSON type	Valid values
sigType	supported RSA signature types — see [FIPS 186-4], Section 5	string	one of {"ansx9.31", "pkcs1v1.5", "pss"}
properties	RSA signature verification parameters — see [FIPS 186-4], Section 5	array	modulo, hashAlg, and saltLen (when sigType is "pss")
modulo	supported RSA modulo for signature verification — see [FIPS 186-4], Section 5	integer	any one of the supported modulo sizes {1024, 2048, 3072, 4096}
hashPair	supported hash algorithms and optional salt length for signature verification for this sigType and modulo# —#see [SP 800-131A], Section 9	array	an array of objects containing a hashAlg and an optional saltLen
hashAlg	supported hash algorithms for this sigType and modulo# —# see [SP 800-131A], Section 9	array	any non-empty subset of {"SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512", "SHA2-512/224", "SHA2-512/256"}
saltLen	supported salt lengths for PSS signature verification — see [FIPS 186-4], Section 5.5	integer	See the note below
NOTE – The 'saltLen' property for each hash algorithm **SHALL** only be present of the 'sigType' is "pss". The values allowed in PSS signature generation is between 0 and the length of the corresponding hash function output block (in bytes), the end points included except for when 1024 is the modulo size. In this case, the maximum 'saltLen' for SHA2-512 is 62 bytes.			

The following is an example of the RSA / sigVer / FIPS186-4 registration

```
{
  "algorithm": "RSA",
  "mode": "sigVer",
  "revision": "FIPS186-4",
  "prereqVals": [
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    },
    {
```

```
      "algorithm": "DRBG",
      "valValue": "654321"
    },
    {
      "algorithm": "SHA",
      "valValue": "7890"
    }
  ],
  "pubExpMode": "random",
  "conformances": [
    "SP800-106"
  ],
  "capabilities": [
    {
      "sigType": "ansx9.31",
      "properties": [
        {
          "modulo": 2048,
          "hashPair": [
            {
              "hashAlg": "SHA2-512"
            }
          ]
        }
      ]
    },
    {
      "sigType": "pkcs1v1.5",
      "properties": [
        {
          "modulo": 4096,
          "hashPair": [
            {
              "hashAlg": "SHA2-224"
            }
          ]
        }
      ]
    },
    {
      "sigType": "pss",
      "properties": [
        {
          "modulo": 3072,
          "hashPair": [
            {
```

```
            "hashAlg": "SHA2-224",
            "saltLen": 28
          }
          {
            "hashAlg": "SHA2-512",
            "saltLen": 64
          }
        ]
      }
    ]
  }
]
}
```

Figure 6

4.7.3. RSA sigVer FIPS186-5 Capabilities Table

The following RSA / sigVer / FIPS186-5 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 11 — Supported RSA sigVer FIPS186-5 Capabilities

JSON value	Description	JSON type	Valid values
sigType	supported RSA signature types — see [FIPS 186-5 (Draft)], Section 5	string	one of {"pkcs1v1.5", "pss"}
properties	RSA signature verification parameters — see [FIPS 186-5 (Draft)], Section 5	array	modulo, hashAlg, and saltLen (when sigType is "pss")
modulo	supported RSA modulo for signature verification — see [FIPS 186-5 (Draft)], Section 5	integer	any one of the supported modulo sizes {2048, 3072, 4096}
maskFunction	the mask function used, only valid for PSS	array	any subset of {"mgf1", "shake-128", "shake-256"}
hashPair	supported hash algorithms and optional salt length for signature verification for this sigType and modulo# —#see [SP 800-131A], Section 9	array	an array of objects containing a hashAlg and an optional saltLen
hashAlg	supported hash algorithms for this sigType and modulo# — see [SP 800-131A], Section 9	array	any non-empty subset of {"SHA-1", "SHA2-224", "SHA2-256", "SHA2-384", "SHA2-512",

JSON value	Description	JSON type	Valid values
			“SHA2-512/224”, “SHA2-512/256”}
saltLen	supported salt lengths for PSS signature verification — see [FIPS 186-5 (Draft)], Section 5.5	integer	See the note below
NOTE – The ‘saltLen’ property for each hash algorithm **SHALL** only be present of the ‘sigType’ is “pss”. The values allowed in PSS signature generation is between 0 and the length of the corresponding hash function output block (in bytes), the end points included.			

See the following abbreviated example for a PSS registration for RSA / sigVer / FIPS186-5.

```
{
  "algorithm": "RSA",
  "mode": "sigVer",
  "revision": "FIPS186-5",
  "prereqVals": [
    {
      "algorithm": "DRBG",
      "valValue": "123456"
    },
    {
      "algorithm": "DRBG",
      "valValue": "654321"
    },
    {
      "algorithm": "SHA",
      "valValue": "7890"
    }
  ],
  "pubExpMode": "random",
  "conformances": [
    "SP800-106"
  ],
  "capabilities": [
    {
      "sigType": "pss",
      "properties": [
        {
          "modulo": 3072,
          "maskFunction": [
            "SHAKE-128", "MGF1"
          ]
          "hashPair": [
```

```
            {
              "hashAlg": "SHA2-224",
              "saltLen": 28
            }
            {
              "hashAlg": "SHA2-512",
              "saltLen": 64
            }
          ]
        }
      ]
    }
  ]
}
```

Figure 7

4.8. RSA SignaturePrimitive Mode Capabilities

The RSA signaturePrimitive mode capabilities (otherwise known as RSASP1 in [RFC 3447]) are advertised as JSON objects within the array of 'capabilities' as part of the 'capability_exchange' element of the ACVP JSON registration message. See the ACVP specification for details on the registration message. In this mode, the only tested capability is the correct exponentiation of 's = msg^d mod n', where 'msg' is a message between '0' and 'n — 1', 'd' is the private exponent and 'n' is the modulus, all supplied by the testing ACVP server. In the event that 'keyFormat' is defined as 'crt', then 'd' is replaced with 'dmp1', 'dmq1', and 'iqmp'. Only 2048-bit RSA keys are allowed for this capability. See Section 5.7 for additional details on constraints for 'msg' and 'n'. See the ACVP specification for details on the registration message.

The following RSA / SignaturePrimitive / 1.0 capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 12 — Supported RSA SignaturePrimitive 1.0 JSON Values

JSON value	Description	JSON type	Valid values
keyFormat	The format by which the client expects the private key to be communicated. Standard refers to the default p, q, d values. Chinese Remainder Theorem uses decomposed values for optimized decryption p, q, dmp1, dmq1, iqmp	string	"standard", "crt"
pubExpMode	Whether the IUT can handle a random or fixed public exponent	string	"random", "fixed"

JSON value	Description	JSON type	Valid values
fixedPubExp	The fixed public exponent e	hex	Any value supported by [FIPS 186-4]: 65537 — 2^256-1, odd

The following is an example of the registration

```
{
    "algorithm": "RSA",
    "mode": "signaturePrimitive",
    "revision": "1.0",
    "keyFormat": "crt",
    "pubExpMode": "fixed",
    "fixedPubExp": "010001"
}
```

Figure 8

4.9. RSA DecryptionPrimitive Mode Capabilities

The RSA decryptionPrimitive mode capabilities are advertised as JSON objects within the array of 'capabilities' as part of the 'capability_exchange' element of the ACVP JSON registration message. A single property is allowed in the registration, 'modulo' with the only approved value of 2048. In this mode, the only tested capability is the correct exponentiation 's = cipherText\^d mod n', where 'cipherText' is a cipherText to be decrypted, 'd' is the private exponent and 'n' is the modulus. See [SP 800-56B], Section 7.1.2 for details.

In testing, only 'cipherText' is supplied by the ACVP server. The client is responsible for generating RSA key pairs of modulus 'n', private key 'd', and calculates 's'. If a client does not support decryption with a standard RSA private exponent 'd', the equivalent Chinese Remainder Theorem (CRT) private key values are allowed to be used. Only 2048-bit RSA keys are allowed for this capability.

See Section 5.9 for additional details on constraints for 'cipherText' and 'n'. The client provides the public exponent 'e', modulus 'n' and the computed result 's' in its response to the ACVP — see Section 6.5. The client must first check if '0 < cipherText < n-1' and return an error if this is not the case. The client returns a value 's' only when 'cipherText' is in the proper range for the size of the selected modulus 'n'. See the ACVP specification for details on the registration message.

An example registration is the following

```
{
    "algorithm": "RSA",
    "mode": "decryptionPrimitive",
    "revision": "1.0",
    "prereqVals":
    [
```

```
        {"algorithm": "DRBG", "valValue": "123456"},
        {"algorithm": "DRBG", "valValue": "654321"},
        {"algorithm": "SHA", "valValue": "7890"}
    ]
}
```

Figure 9

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 crypto algorithm, such as RSA / sigGen / FIPS186-4, RSA / keyVer / FIPS186-5, etc. This section describes the JSON schema for a test vector set used with RSA 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 13 — RSA Vector Set JSON Object

JSON Value	Description	JSON type
acvVersion	Protocol version identifier	string
vsId	Unique numeric identifier for the vector set	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 group JSON objects, which are defined in Section 5.1, Section 5.3, Section 5.5, Section 5.7 or Section 5.9	array

An example of this would look like this

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

Figure 10

5.1. RSA 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. The Test Group JSON object contains meta data that applies to all test vectors within

the group. The following table describes the RSA / keyGen / * JSON elements of the Test Group JSON object.

Table 14 — RSA keyGen Test Group JSON Object

JSON Value	Description	JSON type
modulo	RSA modulus size	integer
hashAlg	the hash algorithm	string
primeTest	Miller-Rabin constraint bound	string
randPQ	keyGen mode used	string
infoGeneratedByServer	Whether or not the test inputs are generated by the server	boolean
keyFormat	The RSA private key format used	string
pubExp	Fixed or random public exponent e	string
testType	Describes the operation the client should perform on the test data	string
tests	Array of individual test cases, see Section 5.2	array

The ‘tgId’, ‘testType’ and ‘tests’ objects **MUST** appear in every test group element communicated from the server to the client as a part of a prompt. Other properties are dependent on which ‘testType’ (see Section 3) the group is addressing.

The impact of the ‘infoGeneratedByServer’ property depends on the other properties identified in the keyGen registration. For ‘randPQs’ of B.3.2, B.3.4, B.3.5, and B.3.6 (or the corresponding FIPS186-5 values), ‘infoGeneratedByServer’ set to true will provide the client with ALL the inputs and expect them to arrive at the same output the server has already arrived at.

In the case of ‘infoGeneratedByServer’ being set to false during the registration, the client will be provided with SOME of the inputs, like an already agreed upon fixed public exponent. However, in general the server will send mostly empty test cases. It is the responsibility of the client to generate both the input properties and output properties comprising the test cases and communicate ALL of those back to the server. The server will try to reach the same output key.

For B.3.3 (or the corresponding FIPS186-5 value of probable), ‘infoGeneratedByServer’ set to true enables the KAT test groups if the client supports a random public exponent. The GDT are always enabled as a bare minimum test for B.3.3 (or probable), independent of the ‘infoGeneratedByServer’ setting.

5.2. RSA 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 RSA / keyGen / * test vector.

The difference between revision "FIPS186-4" and "FIPS186-5" within the prompts lies in the strings used by the server to refer to the various "randPQ", and "primeTest" modes. Revision "FIPS186-4" utilizes the section number from [FIPS 186-4] to refer to the algorithm used while revision "FIPS186-5" utilizes a description of the algorithm.

Table 15 — RSA keyGen Test Case JSON Object

JSON Value	Description	JSON Type
tcId	Test case identifier	integer
e	the public exponent	hex
seed	the seed used in prime generation	hex
bitlens	the length of p1, p2, q1, and q2 for prime generation	array of integers
xP1	xP1 from [FIPS 186-4] B.3.6, step 4	hex
xP2	xP2 from [FIPS 186-4] B.3.6, step 4	hex
xP	the random number used in auxiliary prime generation for p	hex
xQ1	xQ1 from [FIPS 186-4] B.3.6, step 5	hex
xQ2	xQ2 from [FIPS 186-4] B.3.6, step 5	hex
xQ	the random number used in auxiliary prime generation for q	hex
pRand	the random P for testing probable primes according to [FIPS 186-4] or [FIPS 186-5 (Draft)]	hex
qRand	the random Q for testing probable primes according to [FIPS 186-4] or [FIPS 186-5 (Draft)]	hex

Here is an abbreviated yet fully constructed example of the prompt for RSA / keyGen / FIPS186-5

```
{
  "vsId": 2,
  "algorithm": "RSA",
  "mode": "keyGen",
  "revision": "FIPS186-5",
  "testGroups": [
    {
      "tgId": 1,
      "infoGeneratedByServer": true,
```

```
      "modulo": 2048,
      "testType": "AFT",
      "keyFormat": "crt",
      "randPQ": "provable",
      "pubExp": "random",
      "hashAlg": "SHA2-224",
      "tests": [
        {
          "tcId": 1,
          "seed": "5B174CA160...",
          "e": "07D196B84395"
        }
      ]
    },
    {
      "tgId": 3,
      "infoGeneratedByServer": true,
      "modulo": 2048,
      "testType": "AFT",
      "keyFormat": "crt",
      "primeTest": "2powSecStr",
      "randPQ": "probableWithProvableAux",
      "pubExp": "random",
      "tests": [
        {
          "tcId": 7,
          "seed": "B392CFFD8E...",
          "bitlens": [
            142,
            419,
            400,
            334
          ],
          "xp": "F06825A6B...",
          "xq": "BD106DBE5...",
          "e": "3691C632C2BBE7"
        }
      ]
    },
    {
      "tgId": 5,
      "infoGeneratedByServer": false,
      "modulo": 3072,
      "testType": "GDT",
      "keyFormat": "crt",
      "primeTest": "2pow100",
```

```
      "randPQ": "probable",
      "pubExp": "random",
      "tests": [
        {
          "tcId": 13
        }
      ]
    }
  ]
}
```

Figure 11

5.3. RSA 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 RSA / sigGen / * JSON elements of the Test Group JSON object

Table 16 — RSA sigGen Test Group JSON Objects

JSON Values	Description	JSON Type
tgId	Test group identifier	integer
modulo	RSA modulus size	integer
hashAlg	The hash algorithm	string
sigType	Type of signature used in the group	string
saltLen	The salt length for the group in bytes	integer
conformance	Signifies all test cases within the group should utilize random message hashing as described in [SP 800-106]	string
maskFunction	The mask function used for PSS signature scheme	string
testType	Describes the operation the client should perform on the tests data	string
tests	Array of individual test cases, see Section 5.4	array

The ‘tgId’, ‘testType’ and ‘tests’ objects **MUST** appear in every test group element communicated from the server to the client as a part of a prompt. Other properties are dependent on which ‘testType’ (see Section 3) the group is addressing.

NOTE – The ‘maskFunction’ property will only be present for RSA / sigGen / FIPS186-5 inside of test groups for the ‘sigType’ “pss”.

5.4. RSA sigGen Test Cases 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 RSA / sigGen / * test vector.

Table 17 — RSA sigGen Test Case JSON Objects

JSON Values	Description	JSON Type
tcId	Test case idenfitier	integer
message	The message to be signed	hex
saltLen	The length of the salt in bytes	integer
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 ‘saltLen’ property will only be present in test groups for the ‘sigType’ “pss”.		

Here is an abbreviated yet fully constructed example of the prompt for RSA / sigGen / FIPS186-4. The only difference in the structure between RSA / sigGen / FIPS186-4 and RSA / sigGen / FIPS186-5 is the inclusion of the ‘maskFunction’ property in the ‘testGroup’ for RSA / sigGen / FIPS186-5.

```
[
  {
    "acvVersion": " {acvp-version}"
  },
  {
    "vsId": 1163,
    "algorithm": "RSA",
    "mode": "sigGen",
    "revision": "FIPS186-4",
    "testGroups": [
      {
        "tgId": 1,
        "sigType": "ansx9.31",
        "hashAlg": "SHA2-256",
        "modulo": 2048,
        "tests": [
          {
            "tcId": 1165,
            "message": "f648ffc4ed748..."
          }
```

```
        ]
      }
      {
        "tgId": 3,
        "sigType": "pkcs1v1.5",
        "hashAlg": "SHA2-256",
        "modulo": 2048,
        "tests": [
          {
            "tcId": 1167,
            "message": "5af283b1b76ab..."
          }
        ]
      }
      {
        "tgId": 5,
        "sigType": "pss",
        "hashAlg": "SHA2-256",
        "modulo": 2048,
        "tests": [
          {
            "tcId": 1169,
            "saltLen": 20,
            "message": "dfc22604b95d1..."
          }
        ]
      }
    ]
  }
]
```


Figure 12

5.5. RSA 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 RSA / sigVer / * JSON elements of the Test Group JSON object

Table 18 — RSA sigVer Test Group JSON Objects

JSON Values	Description	JSON Type
tgId	Test group identifier	integer
modulo	RSA modulus size	integer

JSON Values	Description	JSON Type
hashAlg	The hash algorithm	string
sigType	Type of signature used in the group	string
saltLen	The salt length for the group in bytes	integer
conformance	Signifies all test cases within the group should utilize random message hashing as described in [SP 800-106]	string
maskFunction	The mask function used for PSS signature scheme	string
n	Public modulus value n for the group	hex
e	Public exponent value e for the group	hex
maskFunction	The mask function used	string
testType	Describes the operation the client should perform on the tests data	string
tests	Array of individual test cases, see Section 6.3	array

The ‘tgId’, ‘testType’ and ‘tests’ objects **MUST** appear in every test group element communicated from the server to the client as a part of a prompt. Other properties are dependent on which ‘testType’ (see Section 3) the group is addressing.

NOTE – The ‘maskFunction’ property will only be present for RSA / sigVer / FIPS186-5 inside of test groups for the ‘sigType’ “pss”.

5.6. Test Cases for sigVer FIPS186-5, FIPS186-4, and FIPS186-2

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 RSA test vector.

Table 19 — RSA sigVer Test Case JSON Objects

JSON Values	Description	JSON Type
tcId	Test case idenfitier	integer
message	The message to be signed	hex
signature	The signature of the message	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

Here is an abbreviated yet fully constructed example of the prompt for RSA / sigVer / FIPS186-4. The only difference in the structure between RSA / sigVer / FIPS186-4 and RSA / sigVer / FIPS186-5 is the inclusion of the ‘maskFunction’ property in the ‘testGroup’ for RSA / sigVer / FIPS186-5.


```
[
  {
    "acvVersion": "{acvp-version}"
  },
  {
    "vsId": 1173,
    "algorithm": "RSA",
    "mode": "sigVer",
    "revision": "FIPS186-4",
    "testGroups": [
      {
        "tgId": 1,
        "sigType": "ansx9.31",
        "hashAlg": "SHA2-256",
        "testType": "AFT",
        "modulo": 2048,
        "e": "166f67",
        "n": "944ded6daaf602e17...",
        "tests": [
          {
            "tcId": 1174,
            "message": "ff17e5e...",
            "signature": "299f1..."
          }
        ]
      },
      {
        "tgId": 4,
        "sigType": "pkcs1v1.5",
        "modulo": 3072,
        "hashAlg": "SHA2-256",
        "testType": "AFT",
        "e": "ac6db1",
        "n": "9bbb099e1ec285594...",
        "tests": [
          {
            "tcId": 1177,
            "message": "921961e...",
            "signature": "55362..."
          }
        ]
```

```
      },
      {
        "tgId": 12,
        "sigType": "pss",
        "modulo": 3072,
        "hashAlg": "SHA2-512",
        "testType": "AFT",
        "e": "fe3079",
        "n": "ce4924ff470fb99d...",
        "conformance": "SP800-106",
        "tests": [
          {
            "tcId": 11179,
            "message": "e49f585...",
            "randomValue": "ab6a9b8b6a75ba76ab76a76b...",
            "randomValueLen": 1024,
            "signature": "4e85f..."
          }
        ]
      }
    ]
  }
]
```


Figure 13

5.7. Test Groups for RSA Signature Primitive 1.0

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 RSA / signaturePrimitive / 1.0 JSON elements of the Test Group JSON object

Table 20 — RSA Signature Primitive 1.0 Test Group Properties

JSON Values	Description	JSON Type
tgId	Test group identifier	integer
testType	Describes the operation the client should perform on the tests data	string
tests	Array of individual test cases	array

The ‘tgId’, ‘testType’ and ‘tests’ objects **MUST** appear in every test group element communicated from the server to the client as a part of a prompt. Other properties are dependent on which ‘testType’ (see Section 3) the group is addressing.

5.8. Test Cases for RSA Signature Primitive 1.0

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 RSA / signaturePrimitive / 1.0 test vector.

Table 21 — RSA Signature Primitive 1.0 Test Case Properties

JSON Values	Description	JSON Type
tcId	Test case idenfitier	integer
n	Modulus	hex
d	Private key exponent, when the keyFormat is “standard”	hex
dmp1	d mod p — 1, when the keyFormat is “crt”	hex
dmq1	d mod q — 1, when the keyFormat is “crt”	hex
iqmp	q\^-1 mod p, when the keyFormat is “crt”	hex
p	prime p, when the keyFormat is “crt”	hex
q	prime q, when the keyFormat is “crt”	hex
message	The message to sign	hex
NOTE 1 – Each test for which ‘message’ is not between ‘0’ and ‘n — 1’ should fail. NOTE 2 – Many private key properties are dependent on the ‘keyFormat’ property from the registration.		

Here is an abbreviated yet fully constructed example of the prompt

```
[
    { "acvVersion": "{acvp-version}" },
    {
        "vsId": 1193,
        "algorithm": "RSA",
        "mode": "signaturePrimitive",
        "revision": "1.0",
        "keyFormat": "standard",
        "testGroups" : [
            {
                "tgId": 1,
                "tests" : [
                    {
                        "tcId" : 1194,
                        "n" : "d0c112f0bee36235d9f...",
                        "d" : "2cde66ea08797aad3cf...",
                        "e": "010001",
                        "message" : "097e82fec7246..."
```

```
                    },
                    {
                        "tcId" : 1195,
                        "n" : "9cd5aa3f0c7c787ee38...",
                        "d" : "0c520729a48d1728ada...",
                        "e" : "010001",
                        "message" : "ffd5aa3f0c7c7..."
                    }
                ]
            }
        ]
    }
]
```

Figure 14

5.9. Test Groups for RSA Decryption Primitive 1.0

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 RSA JSON elements of the Test Group JSON object

Table 22 — RSA Decryption Primitive 1.0 Test Group Properties

JSON Values	Description	JSON Type
tgId	Test group identifier	integer
testType	Describes the operation the client should perform on the tests data	string
totalTests	The total number of elements in the “resultsArray”	integer
totalFailingTests	The number of tests that the client should force to fail in the “resultsArray”	integer
tests	Array of individual test cases	array

The ‘tgId’, ‘testType’ and ‘tests’ objects **MUST** appear in every test group element communicated from the server to the client as a part of a prompt. Other properties are dependent on which ‘testType’ (see Section 3) the group is addressing.

5.10. Test Cases for RSA Decryption Primitive 1.0

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 RSA / DecryptionPrimitive / 1.0 test vector.

Table 23 — RSA Decryption Primitive 1.0 Test Case Properties

JSON Values	Description	JSON Type
tcId	Test case identifier	integer
resultsArray	An array of ciphertexts	array
ciphertext	An individual ciphertext	hex
NOTE – A failing ciphertext is one such that the ciphertext is greater than the n the client provides		

Here is an abbreviated yet fully constructed example of the prompt

```
[
    { "acvVersion": "{acvp-version}" },
    {
        "vsId": 1194,
        "algorithm": "RSA",
        "mode": "decryptionPrimitive",
        "revision": "1.0",
        "testGroups" : [
            {
                "tgId": 1,
                "testType": "AFT",
                "totalTests": 2,
                "totalFailingTests": 1,
                "tests" : [
                    {
                        "tcId" : 1,
                        "resultsArray": [
                            {
                                "cipherText" : "097e82fec72465e..."
                            },
                            {
                                "cipherText" : "ffd5aa3f0c7c787..."
                            }
                        ]
                    }
                ]
            }
        ]
    }
]
```

Figure 15

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 24 — Response JSON Object

JSON Property	Description	JSON Type
acvVersion	The version of the protocol	string
vsId	The vector set identifier	integer
testGroups	The test group data, see Table 25	array

An example of this is the following

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

Figure 16

The testGroups section is used to organize the ACVP client response in a similar manner to how it receives vectors. Several algorithms **SHALL** require the client to send back group level properties in their response. This structure helps accommodate that. The following is a skeleton for the test group structure. Additional properties may be included at this level depending on the algorithm, mode and revision.

Table 25 — Response Test Group JSON Objects

JSON Property	Description	JSON Type
tgId	The test group identifier	integer
tests	The test case data, depending on the algorithm see Table 26, Table 28, Table 30, Table 31 or Table 32	array

An example of this is the following

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

Figure 17

6.1. RSA keyGen Test Group 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 RSA / keyGen / * test vector.

The following table describes the JSON elements for the response to a RSA / keyGen / * test vector.

Table 26 — RSA Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case, unique across the entire vector set	integer
e	the public exponent	hex
testPassed	the verdict on the prime generation testing for the supplied pRand/qRand combination, see [FIPS 186-4], Appendix B.3.3.	boolean
seed	the seed used in prime generation according to [FIPS 186-4], Appendix B. 3.2, B.3.4, or B.3.5	hex
bitlens	the length of p1, p2, q1, and q2 for prime generation according to [FIPS 186-4], Appendix B.3.2, B.3.4, B.3.5 or the length of xP1, xP2, xQ1, and xQ2 for B. 3.6	array of integers
xP1	the prime factor p1 for Primes with Conditions — see [FIPS 186-4], Appendix B.3.3, B.3.4, or B.3.5, if applicable	hex
xP2	the prime factor p2 for Primes with Conditions — see [FIPS 186-4], Appendix B.3.3, B.3.4, or B.3.5, if applicable	hex
xP	the random number used in Step 3 of the algorithm in [FIPS 186-4], Appendix C.9 to generate the prime P, if applicable	hex
p	the private prime factor p	hex
xQ1	the prime factor q1 for Primes with Conditions — see [FIPS 186-4], Appendix B.3.3, B.3.4, or B.3.5, if applicable	hex
xQ2	the prime factor q2 for Primes with Conditions — see [FIPS 186-4],	hex

JSON Value	Description	JSON type
	Appendix B.3.3, B.3.4, or B.3.5, if applicable	
xQ	the random number used in Step 3 of the algorithm in [FIPS 186-4], Appendix C.9 to generate the prime Q, if applicable	hex
q	the private prime factor q	hex
n	the modulus	hex
d	the private exponent d	hex
dmp1	the private exponent d modulo (p — 1) used in a Chinese Remainder Theorem private key	hex
dmq1	the private exponent d modulo (q — 1) used in a Chinese Remainder Theorem private key	hex
iqmp	the multiplicative inverse of q modulo p used in a Chinese Remainder Theorem private key	hex

NOTE 1 – If the 'keyFormat' of the test group is 'standard', then the client **SHALL** not include the 'dmp1', 'dmq1' and 'iqmp' properties. Those properties **SHALL** only be included if the 'keyFormat' is set to 'crt' for the Chinese Remainder Theorem.

NOTE 2 – The 'testPassed' property is only valid for 'KAT' test type groups.

NOTE 3 – If the 'infoGeneratedByServer' test group property is true, then the only response from the client **SHALL** be the values that directly correspond to the key when appropriate, 'p', 'q', 'n', and 'd' (or 'dmp1', 'dmq1', 'iqmp').

NOTE 4 – The 'e' property **SHALL** only be included when the 'infoGeneratedByServer' is false and the 'pubExpMode' is "random".

Use the following applicability grid to determine which properties should be present based on the 'randPQ' test group property. The 'randPQ' property values for RSA / keyGen / FIPS186-4 are based on the section numbers in Appendix B in [FIPS 186-4]. For RSA / keyGen / FIPS186-5, the property values are based on the true names of the generation methods. The RSA / keyGen / FIPS186-5 names will be listed in the grid. This grid only applies to 'AFT' test types.

Table 27 — RSA Test Case Applicability Grid For AFT Responses

JSON Value	Provable	Probable	Provable Primes With Conditions	Provable Primes With Probable Conditions	Probable Primes With Conditions
p	yes	yes	yes	yes	yes
q	yes	yes	yes	yes	yes
e	yes	yes	yes	yes	yes

JSON Value	Provable	Probable	Provable Primes With Conditions	Provable Primes With Probable Conditions	Probable Primes With Conditions
n	yes	yes	yes	yes	yes
d	yes	yes	yes	yes	yes
seed	yes	no	yes	yes	no
xP	no	no	no	yes	yes
xP1	no	no	no	no	yes
xP2	no	no	no	no	yes
xQ	no	no	no	yes	yes
xQ1	no	no	no	no	yes
xQ2	no	no	no	no	yes
bitlens	no	no	yes	yes	yes
NOTE – If the 'crt' key format is used for the group, substitute 'd' with the appropriate values from Table 26.					

The following is an example of an RSA / keyGen / * response. Test group 1 uses the provable prime (B.3.2) generation method. Test group 2 uses the provable prime with conditions (B.3.4) generation method. Test group 3 uses the provable prime with probable conditions (B.3.5) generation method. Test group 4 uses the probable prime with conditions (B.3.6) generation method. Test group 5 is for the 'KAT' test type for probable prime (B.3.3) generation method. Test group 6 uses the probable prime (B.3.3) generation method. In this example, 'infoGeneratedByServer' is set to true.

```
[
  {
    "acvVersion": "<acvp-version>"
  },
  {
    "vsId": 1133,
    "algorithm": "RSA",
    "mode": "keyGen",
    "revision": "FIPS186-4",
    "testGroups": [
      {
        "tgId": 1,
        "tests": [
          {
            "tcId": 1,
            "seed": "5B174CA16001BE8...",
            "n": "8099A2B6C63B2CB2A0...",
            "e": "07D196B84395",
            "p": "B5A06A623B5C7EC4A0...",
            "q": "B5428D256885A767B4...",
            "d": "0A6D3A7F37453EF9EB..."
```

```
      }
    ]
  },
  {
    "tgId": 2,
    "tests": [
      {
        "tcId": 1111,
        "e": "10000021",
        "seed": "af152e46b479af8...",
        "bitlens": [
          312,
          145,
          144,
          338
        ],
        "p": "e2ab16d3026db341223...",
        "q": "d13c3209bbc1bfa27c9...",
        "n": "b942fa09a727ab488f8...",
        "d": "6b56ee657ebf6a54b35..."
      }
    ]
  },
  {
    "tgId": 3,
    "tests": [
      {
        "tcId": 1115,
        "e": "10000021",
        "seed": "e664bc8c8e09ca23...",
        "bitlens": [
          232,
          220,
          336,
          141
        ],
        "xP": "e7b2b10bb6c975ef79...",
        "p": "e7b2b10bb6c975ef794...",
        "xQ": "c3ce8bfcb6fb40bdaf...",
        "q": "c3ce8bfcb6fb40bdafd...",
        "n": "b1380d59234c9f63e63...",
        "d": "bec8baec7da0634211e..."
      }
    ]
  },
  {
```

```
    "tgId": 4,
    "tests": [
      {
        "tcId": 1135,
        "e": "10000021",
        "bitlens": [
          224,
          195,
          352,
          142
        ],
        "xP1": "57c9a2986fc7e69e83...",
        "xP2": "7254d6c998a84230ff...",
        "xP": "c32cccd930ab2c107b3...",
        "p": "c32cccd930ab2c107b3f...",
        "xQ1": "7468d10e69a14b00ec...",
        "xQ2": "20b8c2bae262b13e91...",
        "xQ": "fa97b510539a102879a...",
        "q": "fa97b510539a102879a7...",
        "n": "bf0d69840d0236aa74ea...",
        "d": "166bed3734b922f07446..."
      }
    ]
  },
  {
    "tgId": 5,
    "tests": [
      {
        "tcId": 1119,
        "testPassed": true
      }
    ]
  },
  {
    "tgId": 6,
    "tests": [
      {
        "tcId": 1129,
        "e": "df28ab",
        "p": "e021757c777288dac...",
        "q": "ed1571a9e0cd4a425...",
        "n": "cf91c0065d8e5797f...",
        "d": "1f5201b880a206cb1..."
      }
    ]
  }
```

```
      ]
   }
]
```

Figure 18

6.2. RSA sigGen Test Group 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 RSA / sigGen / * test vector.

The following table describes the JSON elements for the test group responses to a RSA / sigGen / * test vector.

Table 28 — RSA sigGen Test Group Results JSON Object

JSON Value	Description	JSON type
tgId	Numeric identifier for the test group	integer
conformance	Signifies all test cases within the group should utilize random message hashing as described in [SP 800-106]	string
n	The generated modulus for the group	hex
e	The generated public exponent for the group	hex
tests	The individual test cases for the group	array

The following table describes the JSON elements for the test case responses for RSA / sigGen / * .

Table 29 — RSA sigGen Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case	integer
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
signature	The computed signature	hex
NOTE – The 'randomValue' and 'randomValueLen' properties will only be present in test groups for the SP800-106 conformance.		

The following is an example of the response for RSA / sigGen / * .


```
{
  "vsId": 0,
  "algorithm": "RSA",
  "mode": "sigGen",
  "revision": "FIPS186-4",
  "isSample": true,
  "testGroups": [
    {
      "tgId": 1,
      "n": "A31AFF9A3266E5A215C487...",
      "e": "CC288CE6A02AFD",
      "tests": [
        {
          "tcId": 1,
          "signature": "053F1CBA53..."
        },
        {
          "tcId": 2,
          "signature": "04236E8357..."
        },
        {
          "tcId": 3,
          "signature": "4A45C5E696..."
        }
      ]
    },
 {
      "tgId": 2,
      "conformance": "SP800-106",
      "n": "A730E346899141F8B550C...",
      "e": "302D7F30CFCE55",
      "tests": [
        {
          "tcId": 4,
          "randomValue":
 "C44BAE0830813A8E8F989217C967E68A74EDADD9B14128EC877685E6CB631E5F",
          "randomValueLen": 256,
          "signature": "1612B11D9C..."
        },
        {
          "tcId": 5,
          "randomValue":
 "7FBB4BDA3C7C2F67E04583A631000EF783D67ED1BFE7620E9BE897A0270E411C",
          "randomValueLen": 256,
```

```
          "signature": "37387D5C64..."
        },
        {
          "tcId": 6,
          "randomValue":
 "8EF77DED2EB53357BAD523AD16C171221FCF24C0CE15E2FC7F9FBE589A64BEFC",
          "randomValueLen": 256,
          "signature": "4114A93A82..."
        }
      ]
    }
  ]
}
```

Figure 19

6.3. RSA sigVer Test Group 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 RSA / sigVer / * test vector.

The following table describes the JSON elements for the test case responses for RSA / sigVer / * .

Table 30 — RSA sigVer Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case	integer
testPassed	Whether or not the signature provided was valid	boolean

The following is an example of the response for RSA / sigVer / *

```
{
  "vsId": 0,
  "algorithm": "RSA",
  "mode": "sigVer",
  "revision": "FIPS186-4",
  "isSample": true,
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "testPassed": true
        },
```

```
        {
          "tcId": 2,
          "testPassed": false
        },
        {
          "tcId": 3,
          "testPassed": false
        }
      ]
    }
  ]
}
```

Figure 20

6.4. RSA Signature Primitive Test Group 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 RSA / Signature Primitive / * test vector.

The following table describes the JSON elements for the test case responses for RSA / Signature Primitive / * .

Table 31 — RSA Signature Primitive Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case	integer
signature	If the message can be signed, the signature. If the encoded value is shorter than the key modulus it should be padded at the front with zero bytes.	hex
testPassed	If the message could not be signed	boolean

The following is an example of the response for RSA / Signature Primitive / * .

```
{
  "vsId": 0,
  "algorithm": "RSA",
  "mode": "signaturePrimitive",
  "revision": "1.0",
  "isSample": true,
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
```

```
          "tcId": 1,
    "testPassed": true,
          "signature": "a60b879a8fa382fdf4..."
        },
        {
          "tcId": 2,
          "testPassed": false
        },
        {
          "tcId": 3,
          "testPassed": false
        }
      ]
    }
  ]
}
```

Figure 21

6.5. RSA Decryption Primitive Test Group 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 RSA / Decryption Primitive / * test vector.

The following table describes the JSON elements for the test case responses for RSA / Decryption Primitive / * .

Table 32 — RSA Decryption Primitive Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case	integer
e	The public exponent	hex
n	The modulus	hex
plainText	If the ciphertext could be decrypted, the result	hex
testPassed	If the ciphertext could not be decrypted	boolean

The following is an example of the response for RSA / Decryption Primitive / * .

```
{
  "vsId": 0,
  "algorithm": "RSA",
  "mode": "decryptionPrimitive",
  "revision": "1.0",
  "testGroups": [
    {
```

```
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "resultsArray": [
            {
              "e": "60BDBEF656869D",
              "n": "8FA73CF9CAD37456B...",
              "testPassed": false
            },
            {
              "plainText": "009EDAE2D5934F...",
              "e": "D6AA5EF807",
              "n": "A86A73D47F605DCF...",
              "testPassed": true
            }
          ]
        }
      ]
    }
  ]
}
```


Figure 22

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	2019-11-01	Initial Release

Appendix D — References

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

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

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

J. Jonsson, B. Kaliski (February 2003) *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1* (Internet Engineering Task Force), RFC 3447, February 2003. RFC 3447. RFC RFC3447. DOI 10.17487/RFC3447. https://www.rfc-editor.org/info/rfc3447.

National Institute of Standards and Technology (January 2000) *Digital Signature Standard (DSS)* (Gaithersburg, MD), January 2000. FIPS 186-2. https://csrc.nist.gov/publications/detail/fips/186/2/archive/2001-10-05.

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.

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.

Elaine B. Barker, Allen Roginsky (January 2011) *Transitions—Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths* (Gaithersburg, MD), January 2011. SP 800-131A. https://doi.org/10.6028/NIST.SP.800-131A.

Elaine B. Barker, Lily Chen, Andrew R. Regenscheid, Miles E. Smid (August 2009) *Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography* (Gaithersburg, MD), August 2009. SP 800-56B. https://doi.org/10.6028/NIST.SP.800-56B.

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.