Automated Cryptographic Validation Protocol csrc.nist.gov

ACVP ConditioningComponents JSON Specification

Christopher Celi
Information Technology Laboratory
Computer Security Division

September 01, 2020

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

Abstract

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

2. Supported Conditioning Components

Conditioning is an optional process during entropy collection shown in Section 2.2.2 of [SP 800-90B]. There are two types of conditioning components supported by [SP 800-90B]: vetted and non-vetted. A vetted conditioning component comes from a specific list of options. All [SP 800-90B] vetted conditioning components are available via ACVP. This document rounds out the list with options not covered in other algorithm testing.

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

- ConditioningComponents / AES-CBC-MAC / SP800-90B
- ConditioningComponents / BlockCipher_DF / SP800-90B
- ConditioningComponents / Hash_DF / SP800-90B

2.1. Supported Hash Functions for Hash_DF

For the Hash Derivation Function, Hash_DF, the following hash functions **MAY** be advertised by the ACVP compliant cryptographic module:

- SHA-1
- SHA2-224
- SHA2-256
- SHA2-384
- SHA2-512
- SHA2-512/224
- SHA2-512/256

3. Test Types and Test Coverage

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

3.1. Test Types

There is one test-type for Conditioning Components: Algorithm Functional Tests. The testType field definitions are:

- "AFT"—Algorithm Functional Test. These tests can be processed by the client using a normal 'MAC', or 'derive' operation. AFTs cause the implementation under test to exercise nomral operations on a single block, multiple blocks, or partial blocks. In all cases,random data is used. The functional tests are designed to verify that the logical components of the cryptographic implementation (block chunking, block padding etc.) are operating correctly.

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 Conditioning Component 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. Conditioning Component Algorithm Capabilities Registration

This section describes the constructs for advertising support of conditioning component algorithms to the ACVP server.

4.2.1. Block Cipher Based Conditioning Component Capabilities

The following ConditioningComponent / AES-CBC-MAC / SP800-90B and ConditioningComponent / BlockCipher_DF / SP800-90B capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 2 — Block Cipher Conditioning Component Algorithm Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	The algorithm to be validated	string	“ConditioningComponent”
mode	The specific conditioning component to be validated	string	“AES-CBC-MAC” or “BlockCipher_DF”
revision	The algorithm testing revision to use	string	“SP800-90B”
keyLen	The length of keys supported in bits	array	[128, 192, 256]
payloadLen	The lengths in bits supported by the IUT	domain	[{”min”: 8, “max”: 65536, “inc”: 8}]
NOTE – For ConditioningComponent / AES-CBC-MAC / SP800-90B, the payload itself is processed through the encryption engine. Therefore the minimum ‘payloadLen’ is 128 bits and the minimum increment is 128 bits. In other words, all values within the ‘payloadLen’ must correspond to complete AES blocks in bits (a multiple of 128).			

The following is an example of a registration for ConditioningComponents / AES-CBC-MAC / SP800-90B

```
{
  "algorithm": "ConditioningComponent",
  "mode": "AES-CBC-MAC",
  "revision": "SP800-90B",
  "keyLen": [
    128,
    192,
    256
  ],
  "payloadLen": [
    {
      "min": 128,
      "max": 65536,
      "increment": 128
    }
  ]
}
```

Figure 2

The following is an example of a registration for ConditioningComponents / BlockCipher_DF / SP800-90B

```
{
  "algorithm": "ConditioningComponent",
  "mode": "BlockCipher_DF",
  "revision": "SP800-90B",
  "keyLen": [
    128,
    192,
    256
  ],
  "payloadLen": [
    {
      "min": 8,
      "max": 65536,
      "increment": 8
    }
  ]
}
```

Figure 3

4.2.2. Hash Based Conditioning Component Capabilities

The following ConditioningComponent / Hash_DF / SP800-90B capabilities **MAY** be advertised by the ACVP compliant crypto module:

Table 3 — Hash Conditioning Component Algorithm Capabilities JSON Values

JSON Value	Description	JSON type	Valid Values
algorithm	The algorithm to be validated	string	“ConditioningComponent”
mode	The specific conditioning component to be validated	string	“Hash_DF”

JSON Value	Description	JSON type	Valid Values
revision	The algorithm testing revision to use	string	“SP800-90B”
capabilities	An array of supported capability objects	array of objects	Each element in the array is made of exactly one ‘payloadLen’ field and one ‘hashAlg’ field
payloadLen	The lengths in bits supported by the IUT	domain	[{”min”: 1, “max”: 65536, “inc”: 1}]
hashAlg	The hash algorithm that supports the specific lengths	array	Any non-zero number of elements from Section 2.1

The following is an example of a registration for ConditioningComponents / Hash_DF / SP800-90B

```
{
  "algorithm": "ConditioningComponent",
  "mode": "Hash_DF",
  "revision": "SP800-90B",
  "capabilities": [
    {
      "payloadLen": [
        {
          "min": 1,
          "max": 65536,
          "increment": 1
        }
      ],
      "hashAlg": [
        "SHA-1",
        "SHA2-224",
        "SHA2-256",
        "SHA2-384",
        "SHA2-512",
        "SHA2-512/224",
        "SHA2-512/256"
      ]
    }
  ]
}
```

Figure 4

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 Conditioning Component algorithms.

The test vector set JSON schema is a multi-level hierarchy that contains meta data for the entire vector set as well as individual test vectors to be processed by the ACVP client. The following table describes the JSON elements at the top level of the hierarchy.

Table 4 — Top Level Test Vector JSON Elements

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

An example of this would look like this

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

Figure 5

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 ConditioningComponent / AES-CBC-MAC / SP800-90B, ConditioningComponent / Hash_DF / SP800-90B, etc. This section describes the JSON schema for a test vector set used with Conditioning Component crypto algorithms.

The test vector set JSON schema is a multi-level hierarchy that contains meta data for the entire vector set as well as individual test vectors to be processed by the ACVP client. The following table describes the JSON elements at the top level of the hierarchy.

Table 5 — Conditioning Component 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, Section 6.3, or Section 6.5 depending on the algorithm	array

6.1. Conditioning Component AES-CBC-MAC 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 ConditioningComponent / AES-CBC-MAC / SP800-90B JSON elements of the Test Group JSON object.

Table 6 — Conditioning Component AES-CBC-MAC Test Group JSON Object

JSON Value	Description	JSON type
tgId	The unique group identifier	integer
testType	Describes the operation the client should perform on the test data	string
keyLen	The length of the key used in the group	integer
tests	Array of individual test cases, see Section 6.2	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.

6.2. Conditioning Component AES-CBC-MAC 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 ConditioningComponent / AES-CBC-MAC / SP800-90B test vector.

Table 7 — Conditioning Component AES-CBC-MAC Test Case JSON Object

JSON Value	Description	JSON Type
tcId	Test case identifier	integer
pt	The plaintext	hex
key	The key	hex

Here is an abbreviated yet fully constructed example of the prompt for ConditioningComponent / AES-CBC-MAC / SP800-90B

```
{
  "vsId": 42,
  "algorithm": "ConditioningComponent",
  "mode": "AES-CBC-MAC",
  "revision": "SP800-90B",
  "testGroups": [
    {
      "tgId": 1,
      "testType": "AFT",
      "keyLen": 128,
      "tests": [
        {
          "tcId": 1,
          "pt": "FE44418EF94E5DA8...",
          "key": "E618ADF7E7CEBB46465C0B18A924768A"
        },
        {
          "tcId": 2,
          "pt": "6ABEED30F813C137D47BF1E9E837DAEE",
          "key": "D1C1B7FFB2CCE0BBF13D4F7B4A246A8D"
        }
      ]
    }
  ]
}
```

Figure 6

6.3. Conditioning Component BlockCipher_DF 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 ConditioningComponent / BlockCipher_DF / SP800-90B JSON elements of the Test Group JSON object.

Table 8 — Conditioning Component BlockCipher_DF Test Group JSON Object

JSON Value	Description	JSON type
tgId	The unique group identifier	integer
testType	Describes the operation the client should perform on the test data	string
keyLen	The length of the key used in the group	integer
tests	Array of individual test cases, see Section 6.4	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.

6.4. Conditioning Component BlockCipher_DF 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 ConditioningComponent / BlockCipher_DF / SP800-90B test vector.

Table 9 — Conditioning Component BlockCipher_DF Test Case JSON Object

JSON Value	Description	JSON Type
tcId	Test case identifier	integer
payload	The input into the derivation function	hex
payloadLen	The length in bits of the input	integer

Here is an abbreviated yet fully constructed example of the prompt for ConditioningComponent / BlockCipher_DF / SP800-90B

```
{
  "vsId": 42,
  "algorithm": "ConditioningComponent",
  "mode": "BlockCipher_DF",
  "revision": "SP800-90B",
  "testGroups": [
    {
      "tgId": 1,
      "keyLen": 128,
      "testType": "AFT",
      "tests": [
        {
          "tcId": 1,
          "payload": "2874215320DADAC...",
          "payloadLen": 54112
        },
        {
          "tcId": 2,
          "payload": "36",
          "payloadLen": 8
        }
      ]
    }
  ]
}
```

Figure 7

6.5. Conditioning Component Hash_DF 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 ConditioningComponent / Hash_DF / SP800-90B JSON elements of the Test Group JSON object.

Table 10 — Conditioning Component Hash_DF Test Group JSON Object

JSON Value	Description	JSON type
tgId	The unique group identifier	integer
testType	Describes the operation the client should perform on the test data	string
hashAlg	The hash algorithm used in the derivation function	string
tests	Array of individual test cases, see Section 6.6	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.

6.6. Conditioning Component Hash_DF 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 ConditioningComponent / Hash_DF / SP800-90B test vector.

Table 11 — Conditioning Component Hash_DF Test Case JSON Object

JSON Value	Description	JSON Type
tcId	Test case identifier	integer
payload	The input into the derivation function	hex
payloadLen	The length in bits of the input	integer

Here is an abbreviated yet fully constructed example of the prompt for ConditioningComponent / Hash_DF / SP800-90B

```json
{
  "vsId": 42,
  "algorithm": "ConditioningComponent",
  "mode": "Hash_DF",
  "revision": "SP800-90B",
  "testGroups": [
    {
      "tgId": 1,
      "hashAlg": "SHA2-256",
      "testType": "AFT",
      "tests": [
        {
          "tcId": 1,
          "payload": "2874215320DADAC...",
          "payloadLen": 54112
        },
        {
          "tcId": 2,
          "payload": "36",
          "payloadLen": 8
        }
      ]
    }
  ]
}
```

Figure 8

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 12 — 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 13	array

An example of this is the following

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

Figure 9

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 13 — 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 14, Table 15, or Table 16	array

An example of this is the following

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

Figure 10

7.1. Conditioning Component AES-CBC-MAC Test 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 ConditioningComponent / AES-CBC-MAC / SP800-90B test vector.

The following table describes the JSON elements for the test case responses for ConditioningComponent / AES-CBC-MAC / SP800-90B.

Table 14 — Conditioning Component AES-CBC-MAC Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case	integer
ct	The ciphertext output	hex
NOTE – In the case of AES-CBC-MAC, the output is always 128-bits regardless of the size of the input.		

The following is an example of the response for ConditioningComponent / AES-CBC-MAC / SP800-90B .

```
{
  "vsId": 42,
  "algorithm": "ConditioningComponent",
  "mode": "AES-CBC-MAC",
  "revision": "SP800-90B",
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "ct": "4A8575F3EA300812C60B19678620CA9F"
        },
        {
          "tcId": 2,
          "ct": "2F85CD9748F4CEE2F9BAE939874D8321"
        }
      ]
    }
  ]
}
```

Figure 11

7.2. Conditioning Component BlockCipher_DF Test 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 ConditioningComponent / BlockCipher_DF / SP800-90B test vector.

The following table describes the JSON elements for the test case responses for ConditioningComponent / BlockCipher_DF / SP800-90B.

Table 15 — Conditioning Component BlockCipher_DF Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case	integer
requestedBits	The output of the derivation function	hex
NOTE – In the case of BlockCipher_DF, the output is always 128-bits regardless of the size of the input.		

The following is an example of the response for ConditioningComponent / BlockCipher_DF / SP800-90B .

```
{
  "vsId": 42,
  "algorithm": "ConditioningComponent",
  "mode": "BlockCipher_DF",
  "revision": "SP800-90B",
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "requestedBits": "4A8575F3EA300812C60B19678620CA9F"
        },
        {
          "tcId": 2,
          "requestedBits": "2F85CD9748F4CEE2F9BAE939874D8321"
        }
      ]
    }
  ]
}
```

Figure 12

7.3. Conditioning Component Hash_DF Test 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 ConditioningComponent / Hash_DF / SP800-90B test vector.

The following table describes the JSON elements for the test case responses for ConditioningComponent / Hash_DF / SP800-90B.

Table 16 — Conditioning Component Hash_DF Test Case Results JSON Object

JSON Value	Description	JSON type
tcId	Numeric identifier for the test case	integer
requestedBits	The output of the derivation function	hex

The following is an example of the response for ConditioningComponent / Hash_DF / SP800-90B .

```
{
  "vsId": 42,
  "algorithm": "ConditioningComponent",
  "mode": "Hash_DF",
  "revision": "SP800-90B",
  "testGroups": [
    {
      "tgId": 1,
      "tests": [
        {
          "tcId": 1,
          "requestedBits": "4A8575F3EA300812C60B19678620CA9F"
        },
        {
          "tcId": 2,
          "requestedBits": "2F85CD9748F4CEE2F9BAE939874D8321"
        }
      ]
    }
  ]
}
```

Figure 13

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	2020-09-01	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.

Dr. Meltem Sönmez Turan, Elaine B. Barker, John M. Kelsey, Kerry A. McKay, Mary L. Baish, Michael Boyle (January 2018) *Recommendation for the Entropy Sources Used for Random Bit Generation* (Gaithersburg, MD), January 2018. SP 800-90B. https://doi.org/10.6028/NIST.SP.800-90B.

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.