ACVP TLS Key Derivation Function JSON Specification

Russell Hammett
HII Technical Solutions Division
302 Sentinel Drive, Suite #300, Annapolis Junction, MD 20701

June 05, 2019

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

Abstract

This document defines the JSON schema for testing RFC8446 TLS v1.3 KDF 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 RFC8446 TLS v1.3 KDF 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 RFC8446 TLS v1.3 KDF implementations using ACVP.

2. Supported KDFs

The following key derivation functions **MAY** be advertised by the ACVP compliant cryptographic module:

- TLS-v1.3 / KDF / RFC8446

3. Test Types and Test Coverage

This section describes the design of the tests used to validate RFC8446 TLS v1.3 KDF implementations.

3.1. Test Types

There is only one test type: functional tests. Each has a specific value to be used in the testType field. The testType field definition is:

- "AFT" — Algorithm Functional Test. These tests can be processed by the client using a normal 'derive_key' operation. AFTs cause the implementation under test to exercise normal 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 key deriviation process are operating correctly.

3.2. Test Coverage

The tests described in this document have the intention of ensuring an implementation is conformant to [RFC 8446].

3.2.1. Requirements Covered

- RFC8446 — 7.1 Key Schedule. This section of the document encapsulates the bulk of ACVP testing; any functions involved in the computation of the TLS v1.3 KDF **SHALL BE** tested as a part of the vector set. Note that message construction that is defined by the protocol for messages such as ClientHello, ServerHello, etc **SHALL NOT** be within the scope of ACVP testing.
- RFC8446 — 4.4.1 The Transcript Hash. The transcript hash is used within section 7.1 for deriving keying material and as such **SHALL BE** tested as a part of ACVP testing.

3.2.2. Requirements Not Covered

- RFC8446 — ACVP testing for the TLS v1.3 KDF is done to ensure proper implementation of the KDF portion of the RFC; as such TLS protocol specific pieces are not tested. Protocol specific construction of messages such as the ClientHello, ServerHello, etc **SHALL NOT** be within the scope of ACVP 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 TLS v1.3 KDF 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. Property Registration

4.3. Registration Example

A registration **SHALL** use these properties

Table 2 — TLS Registration JSON Values

JSON Property	Description	JSON Type	Valid Values
algorithm	Name of the algorithm to be validated	string	“TLS-v1.3”
mode	Mode of the algorithm to be validated	string	“KDF”
revision	ACVP Test version	string	“RFC8446”
hmacAlg	HMAC functions supported	array	See Section 4.3.1
runningMode	The supported key exchange modes for the KDF https://tools.ietf.org/html/rfc8446#section-2	array	See Section 4.3.2

4.3.1. Valid HMAC Functions

The following hash functions **MAY** be advertised by an ACVP compliant client under the ‘hmacAlg’ property

- SHA2-256
- SHA2-384

4.3.2. Valid Running Modes

The KDF modes **MAY** be advertised by an ACVP compliant client under the ‘runningMode’ property

- PSK
- DHE
- PSK-DHE

An example registration within an algorithm capability exchange looks like this

```
{
  "algorithm": "TLS-v1.3",
  "mode": "KDF",
  "revision": "RFC8446",
  "hmacAlg": [
    "SHA2-256",
    "SHA2-384",
```

```
    ],
  "runningMode": [
    "PSK-DHE"
  ]
}
```

Figure 2

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 RFC8446 TLS v1.3 KDF 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 3 — 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 5.1	array

An example of this would look like this

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

Figure 3

5.1. Test Groups

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 RFC8446 TLS v1.3 KDF JSON elements of the Test Group JSON object

Table 4 — Test Group JSON Object

JSON Value	Description	JSON Type
tgId	Test group identifier	integer
testType	Test operations to be performed	string
hmacAlg	SHA version used	string
runningMode	The key exchange mode used	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.

5.2. Test Cases

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 RFC8446 TLS v1.3 KDF test vector.

Table 5 — Test Case JSON Object

JSON Value	Description	JSON Type
tcId	Test case identifier	integer
psk	Random pre-shared key, included for PSK and PSK-DHE running modes.	hex
dhe	Random Diffie-Hellman shared secret, included for DHE and PSK-DHE running modes.	hex
helloClientRandom	Randomly generated Client Hello message	hex
helloServerRandom	Randomly generated Server Hello message	hex
finishedClientRandom	Randomly generated Client Finished message	hex
finishedServerRandom	Randomly generated Server Finished message	hex

Note that when the PSK or DHE are not included in the test case, it is assumed they are the digest size of the group’s hash set to zero. As an example, for a test group using SHA2-256, with a running mode of “DHE”, the “PSK” would be represented by a BitString of 256 bits or 32 bytes, all being “0”.

Here is an abbreviated yet fully constructed example of the prompt.

```
{
```

```
    "vsId": 1,
    "algorithm": "TLS-v1.3",
    "mode": "KDF",
    "revision": "RFC8446",
    "testGroups": [
        {
            "tgId": 1,
            "testType": "AFT",
            "hmacAlg": "SHA2-256",
            "runningMode": "PSK-DHE",
            "tests": [{
                "tcId": 1,
                "psk":
 "C2A39E5D172C7D147B5FD3752E4C0840EDDF7C5684B00E7B1AA20B7F56CF64F1EE05",
                "dhe":
 "EE6ADDBC8BDF014254051D78D7A0ECE35AAB230024647E871B375C257E23FC814235",
                "helloClientRandom":
 "7613AA5EAC7D233CCFF764C95D22B6BB026C087DF017E04C5AF6F0E2C1C9E3FA134F",
                "helloServerRandom":
 "A1CDA1FC4A72BF85FAEC964AF1CBB421BA1FD683513C24D11EE0B0092190C729ECD4",
                "finishedClientRandom":
 "F77C6488E63463FEDACA4CEF38F66E1957BDCB1F9A6719029F75590687487AF7F235",
                "finishedServerRandom":
 "9B562F22963D4A9C96DA93A1E3AA5F88A48CA7A0EA92D293AA7C72AD9A71A4DF911B"
            }]
        }
    ]
}
```


Figure 4

6. 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 6 — Vector Set 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	array

An example of this is the following

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

Figure 5

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.

Table 7 — Vector Set Group Response JSON Object

JSON Property	Description	JSON Type
tgId	The test group identifier	integer
tests	The test case data	array

An example of this is the following

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

Figure 6

The following table describes the JSON object that represents a test case response for a RFC8446 TLS v1.3 KDF.

Table 8 — Test Case Results JSON Object

JSON Property	Description	JSON Type
tcId	The test case identifier	integer

JSON Property	Description	JSON Type
clientEarlyTrafficSecret	The client early traffic secret. `Derive-Secret(., "c e traffic", ClientHello)`	hex
earlyExporterMasterSecret	The early exporter master secret. `Derive-Secret(., "e exp master", ClientHello)`	hex
clientHandshakeTrafficSecret	The client handshake traffic secret. `Derive-Secret(., "c hs traffic", ClientHello… ServerHello)`	hex
serverHandshakeTrafficSecret	The server handshake traffic secret. `Derive-Secret(., "s hs traffic", ClientHello… ServerHello)`	hex
clientApplicationTrafficSecret	The client application traffic secret. `Derive-Secret(., "c ap traffic", ClientHello… server Finished)`	hex
serverApplicationTrafficSecret	The server application traffic secret. `Derive-Secret(., "s ap traffic", ClientHello… server Finished)`	hex
exporterMasterSecret	The exporter master secret. `Derive-Secret(., "exp master", ClientHello… server Finished)`	hex
resumptionMasterSecret	The resumption master secret. `Derive-Secret(., "res master",`	hex

JSON Property	Description	JSON Type
	ClientHello… client Finished)	

Here is an abbreviated example of the response

```
{
  "vsId": 1,
  "testGroups": [{
    "tgId": 1,
    "tests": [{
      "tcId": 1,
      "clientEarlyTrafficSecret":
 "6C4A2365493D1AF0F654D59A181EDD2510205F6AE3F22B6EE765B8208C99C66C",
      "earlyExporterMasterSecret":
 "3C7204D21F2C10CD744915221CD46B8CF914E5E3124C87A822959C4EB58F13B9",
      "clientHandshakeTrafficSecret":
 "C017D71DB97422E10BF950C1B530AB754A11D4FCF562DDBB47EE6DBAAD72CBA3",
      "serverHandshakeTrafficSecret":
 "42D17CF61E98CA182AF677E13D40EA513627D950156BC8D80E749AB789930DE7",
      "clientApplicationTrafficSecret":
 "52E4E879CA4AA2314AE5F2CCF3870BA879AE2644DB96BEC71493ADEAEE8EA121",
      "serverApplicationTrafficSecret":
 "D4D3C87A5B5F165E90807BA7954734BA733668E9858CCE3F34B1A3E4965681F8",
      "exporterMasterSecret":
 "7880C0A3BC51A1B2FE4013DB481B17B3D0A1DC6C3688EB178BF6FCD3D306AE9A",
      "resumptionMasterSecret":
 "753DB3A1743DAA17AE52E2B6AA6E00AA07DA46111A43653325C98D69079E7BF4"
    }]
  }]
}
```

Figure 7

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	2019-06-05	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.

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.

E. Rescorla (August 2018) *The Transport Layer Security (TLS) Protocol Version 1.3* (Internet Engineering Task Force), RFC 8446, August 2018. RFC 8446. RFC RFC8446. DOI 10.17487/RFC8446. https://www.rfc-editor.org/info/rfc8446.

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.