Test Vectors for Kyber and Dilithium

Test Vectors

<series>draft-hallambaker-test-vectors

<status>informational

<stream>independent

<ipr>trust200902

<author>Phillip Hallam-Baker

<surname>Hallam-Baker

<initials>P. M.

<firstname>Phillip

<email>phill@hallambaker.com

An extended set of test vectors for the Kyber and Dilithium algorithms is presented. These are intended to allow implementers to ensure that their implementations exercise all the code paths in the algorithms.

# Introduction

The structure of post quantum algorithms differs from traditional public key algorithms in ways that make traditional approaches to presenting test vectors unsatisfactory. Public and private keys are considerably larger than for existing algorithms and the structure of the algorithms themselves is considerably more complex.

While presenting a single set of test points per algorithm has proved sufficient for algorithms such as RSA and Ed448, this approach does not provide the same degree of assurance for Kyber or Dilithium unless the test vectors themselves are chosen with great care.

This document presents a set of test vectors for the Kyber and Dilithium algorithms which may be of assistance to implementers looking to verify their implementations in a repeatable fashion.

Providing test vector information in compressed form allows a broader set of outputs to be provided.

## Deterministic Seed Generation

Kyber and Dilithium require a source of secret random information. These are used to seed key generation in both algorithms, to seed the key exchange mechanism in Kyber and to generate a random input text for signature in Dilithium.

For interoperability and conformance testing purposes, it is desirable to replace these random seed values with fixed values to provide repeatability.

Traditionally, such seed values have been specified as byte sequences. But entry of byte sequences introduces a possible source of error as implementation languages have different requirements for entry of data constants.

Test vector labels provide a compact means of specifying test vectors of arbitrary size in a convenient form. For example:

Kyber-1024-0-Keygen:32

Kyber-1024-0-Session:32

Dilithium-Mode5-0-Keygen:32

Dilithium-Mode5-0-Message:128

The test vector label Kyber-1024-0-Keygen:32 requires little explanation, it is a test vector of 32 bytes generated to test the Kyber algorithm at 1024 bit strength. The number 0 refers to the test number and the tag ‘Keygen’ specifies that the seed is used for key generation.

Since both algorithms make extensive use of SHA3, it is convenient to use SHAKE256 to convert the test vector label to the corresponding byte sequence.

To convert the label Kyber-1024-0-Keygen:32 to a byte sequence, we convert the text sequence Kyber-1024-0-Keygenh:32 to a byte sequence and perform a SHAKE256 operation on it to extract the required 32 bytes.

By design, the test vector labels make use of the ASCII character set so that the use of ASCII or UTF8 encoding produces the same result.

The approach to test vector labeling provides a convenient means of referring to each test.

## Result Fingerprints

Result fingerprints provide a convenient means of presenting algorithm outputs. A result fingerprint consists of the SHA-3-256 digest of the algorithm output presented in base 16 with separator characters every four characters.

Result fingerprints may be truncated to the first 32 or first 128 bits of output to aid comprehension.

## Byte Sequence Abstracts

Byte Sequence abstracts provide a means of presenting the most critical parts of a data output without presenting the entire output. This allows programming errors related to data sizes, alignments etc. to be quickly identified.

0001-0203— [24] -2829-3031

Byte Sequence abstracts may also be used to specify cases in which a data structure is varied for testing purposes. For example, to corrupt an input to test input validation or to check that verification fails if a signature is modified. In these cases, it is sufficient to specify only the modified bytes.

# Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in <norm="RFC2119"/>.

## Defined Terms

## Related Specifications

## Implementation Status

# Kyber

## Key Generation

## Encryption

## Decryption

# Dilithium

## Key Generation

## Signature

## Verification

# Security Considerations

# IANA Considerations

This document does not create any actions for IANA

# Acknowledgements

# Appendix A: Kyber Unredacted

# Appendix B: Dilithium Unredacted