Stream: Internet Engineering Task Force (IETF)
RFC: 9814
Category: Standards Track
Published: July 2025
ISSN: 2070-1721
Authors: R. Housley, *Vigil Security*; S. Fluhrer, *Cisco Systems*; P. Kampanakis, *Amazon Web Services*; B. Westerbaan, *Cloudflare*

RFC 9814
Use of the SLH-DSA Signature Algorithm in the Cryptographic Message Syntax (CMS)

Abstract

SLH-DSA is a stateless hash-based signature algorithm. This document specifies the conventions for using the SLH-DSA signature algorithm with the Cryptographic Message Syntax (CMS). In addition, the algorithm identifier and public key syntax are provided.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9814.

Table of Contents

1. Introduction

This document specifies the conventions for using the SLH-DSA hash-based signature algorithm [FIPS205] with the Cryptographic Message Syntax (CMS) [RFC5652] signed-data content type.

SLH-DSA offers two signature modes: pure mode and pre-hash mode. SLH-DSA signature operations include a context string as input. The context string has a maximum length of 255 bytes. By default, the context string is the empty string. This document only specifies the use of pure mode with an empty context string for the CMS signed-data content type.

SLH-DSA offers three security levels. The parameters for each of the security levels were chosen to provide 128 bits of security, 192 bits of security, and 256 bits of security. Separate algorithm identifiers have been assigned for SLH-DSA at each of these security levels.

SLH-DSA is a stateless hash-based signature algorithm. Other hash-based signature algorithms are stateful, including Hierarchical Signature System (HSS) / Leighton-Micali Signatures (LMS) [RFC8554] and eXtended Merkle Signature Scheme (XMSS) [RFC8391]. Without the need for state kept by the signer, SLH-DSA is much less fragile than the stateful hash-based signature algorithms.

1.1. ASN.1

CMS values are generated with ASN.1 [X680] using the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X690].

1.2. Motivation

There have been recent advances in cryptanalysis and advances in the development of quantum computers. Each of these advances pose a threat to widely deployed digital signature algorithms.

If Cryptographically Relevant Quantum Computers (CRQCs) are ever built, they will be able to break many of the public key cryptosystems currently in use, including RSA, DSA, Elliptic Curve Digital Signature Algorithm (ECDSA), and Edwards-curve Digital Signature Algorithm (EdDSA). A Post-Quantum Cryptosystem (PQC) is secure against quantum computers that have more than a trivial number of quantum bits (qu-bits). It is open to conjecture when it will be feasible to build such quantum computers; however, it is prudent to use cryptographic algorithms that remain secure if a CRQC is invented. SLH-DSA is a PQC signature algorithm.

1.3. Terminology

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. SLH-DSA Hash-Based Signature Algorithm Overview

SLH-DSA is a stateless hash-based signature algorithm. SLH-DSA makes use of a few time signature constructions, namely Forest of Random Subsets (FORS) and a hypertree. FORS signs a message with a private key. The corresponding FORS public keys are the leaves in k binary trees. The roots of these trees are hashed together to form a FORS root. SLH-DSA uses a one-time signature scheme called Winternitz One-Time Signature Plus (WOTS+). The FORS tree roots are signed by a WOTS+ private key. The corresponding WOTS+ public keys form the leaves in d-layers of Merkle subtrees in the SLH-DSA hypertree. The bottom layer of that hypertree signs the FORS roots with WOTS+. The roots of the bottom Merkle subtrees are then signed with WOTS+ and the corresponding WOTS+ public keys form the leaves of the next-level-up subtree. Subtree roots are consequently signed by their corresponding subtree layers until the top subtree is reached. The top-layer subtree forms the hypertree root, which is trusted at the verifier.

An SLH-DSA signature consists of the randomization string, the FORS signature, the WOTS+ signature in each layer, and the path to the root of each subtree until the root of the hypertree is reached.

An SLH-DSA signature is verified using the FORS signature, the WOTS+ signatures, and the path to the root of each subtree. When reaching the root of the hypertree, the signature verifies only if it hashes to the pre-trusted root of the SLH-DSA hypertree.

SLH-DSA is a stateless hash-based signature algorithm. Stateful hash-based signature schemes require that the WOTS+ private key (generated by using a state index) never be reused or the scheme loses its security. FORS is used at the bottom of the SLH-DSA hypertree. Security decreases if the same private key is used to sign two or more different messages, but security does not collapse, as is the case in stateful hash-based signature algorithms. Without the need for state kept by the signer to ensure it is not reused, SLH-DSA is much less fragile.

SLH-DSA was designed to sign up to 2^{64} messages and offers three security levels. The parameters of the SLH-DSA hypertree include the security parameter, the hash function, the tree height, the number of layers of subtrees, the Winternitz parameter of WOTS+, and the number of FORS trees and leaves in each. The parameters for each of the security levels were chosen to be at least as secure as a generic block cipher of 128, 192, or 256 bits.

3. SLH-DSA Public Key Identifier

The AlgorithmIdentifier for an SLH-DSA public key **MUST** use one of the twelve id-slh-dsa object identifiers listed below, based on the security level used to generate the SLH-DSA hypertree, the small or fast version of the algorithm, and the use of SHA2 [FIPS180] or SHAKE [FIPS202]. For example, id-slh-dsa-shake-256s represents the 256-bit security level, the small version of the algorithm, and the use of SHAKE256. The parameters field of the AlgorithmIdentifier for the SLH-DSA public key **MUST** be absent.

```
nistAlgorithms OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
  country(16) us(840) organization(1) gov(101) csor(3) 4 }

sigAlgs OBJECT IDENTIFIER ::= { nistAlgorithms 3 }

id-slh-dsa-sha2-128s OBJECT IDENTIFIER ::= { sigAlgs 20 }

id-slh-dsa-sha2-128f OBJECT IDENTIFIER ::= { sigAlgs 21 }

id-slh-dsa-sha2-192s OBJECT IDENTIFIER ::= { sigAlgs 22 }

id-slh-dsa-sha2-192f OBJECT IDENTIFIER ::= { sigAlgs 23 }

id-slh-dsa-sha2-256s OBJECT IDENTIFIER ::= { sigAlgs 24 }

id-slh-dsa-sha2-256f OBJECT IDENTIFIER ::= { sigAlgs 25 }

id-slh-dsa-shake-128s OBJECT IDENTIFIER ::= { sigAlgs 26 }

id-slh-dsa-shake-128f OBJECT IDENTIFIER ::= { sigAlgs 27 }

id-slh-dsa-shake-192s OBJECT IDENTIFIER ::= { sigAlgs 28 }

id-slh-dsa-shake-192f OBJECT IDENTIFIER ::= { sigAlgs 29 }

id-slh-dsa-shake-256s OBJECT IDENTIFIER ::= { sigAlgs 30 }

id-slh-dsa-shake-256f OBJECT IDENTIFIER ::= { sigAlgs 31 }
```

When this AlgorithmIdentifier appears in the SubjectPublicKeyInfo field of an X.509 certificate [RFC5280], the certificate key usage extension **MAY** contain digitalSignature, nonRepudiation, keyCertSign, and cRLSign; the certificate key usage extension **MUST NOT** contain other values.

```
    pk-slh-dsa-sha2-128s PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-sha2-128s
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-sha2-128f PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-sha2-128f
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-sha2-192s PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-sha2-192s
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-sha2-192f PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-sha2-192f
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-sha2-256s PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-sha2-256s
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-sha2-256f PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-sha2-256f
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-shake-128s PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-shake-128s
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-shake-128f PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-shake-128f
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-shake-192s PUBLIC-KEY ::= {
```

```
        IDENTIFIER id-slh-dsa-shake-192s
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-shake-192f PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-shake-192f
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-shake-256s PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-shake-256s
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    pk-slh-dsa-shake-256f PUBLIC-KEY ::= {
        IDENTIFIER id-slh-dsa-shake-256f
        -- KEY no ASN.1 wrapping --
        CERT-KEY-USAGE
          { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
        -- PRIVATE-KEY no ASN.1 wrapping -- }

    SLH-DSA-PublicKey ::= OCTET STRING (SIZE (32 | 48 | 64))

    SLH-DSA-PrivateKey ::= OCTET STRING (SIZE (64 | 96 | 128))
```

No additional encoding of the SLH-DSA public key is applied in the SubjectPublicKeyInfo field of an X.509 certificate [RFC5280].

No additional encoding of the SLH-DSA private key is applied in the PrivateKeyInfo field of the privateKey field of the OneAsymmetricKey type of an Asymmetric Key Package [RFC5958].

When an SLH-DSA public key appears outside of a SubjectPublicKeyInfo type in an environment that uses ASN.1 encoding, the SLH-DSA public key can be encoded as an OCTET STRING by using the SLH-DSA-PublicKey type.

When an SLH-DSA private key appears outside of an Asymmetric Key Package in an environment that uses ASN.1 encoding, the SLH-DSA private key can be encoded as an OCTET STRING by using the SLH-DSA-PrivateKey type.

4. Signed-Data Conventions

As specified in CMS [RFC5652], the digital signature is produced from the message digest and the signer's private key. The signature is computed over different values depending on whether signed attributes are absent or present.

When signed attributes are absent, the SLH-DSA (pure mode) signature is computed over the content. When signed attributes are present, a hash **MUST** be computed over the content using the same hash function that is used in the SLH-DSA tree. The signed attributes **MUST** include a content-type attribute and a message-digest attribute. The message-digest attribute contains the hash value of the content. The SLH-DSA signature is computed over the DER encoding of the set of signed attributes. The SLH-DSA signature-generation operation is called slh_sign; see Section 10.2.1 of [FIPS205]. In summary:

```
IF (signed attributes are absent)
THEN slh_sign(content)
ELSE message-digest attribute = Hash(content);
     slh_sign(DER(SignedAttributes))
```

In some implementations, performance may be significantly improved by signing and verifying DER(SignedAttributes) when the content is large. That is, passing an entire large message content to the signing function or the signature validation function can have an impact on performance. When the signed attributes are present, Section 5.3 of [RFC5652] requires the inclusion of the content-type attribute and the message-digest attribute. Other attributes can also be included.

When using SLH-DSA and signed attributes are present in the SignerInfo, the digestAlgorithms field in the SignedData **MUST** include the identifier for the one-way hash function used to compute the message digest.

When using SLH-DSA, the fields in the SignerInfo are used as follows:

digestAlgorithm:
: The digestAlgorithm **MUST** identify a one-way hash function. When signed attributes are absent, the digestAlgorithm identifier **MUST** match the hash function used in the SLH-DSA tree (as shown in the list below). When signed attributes are present, to ensure collision resistance, the identified hash function **MUST** produce a hash value that is at least twice the size of the hash function used in the SLH-DSA tree. The hash functions defined in [FIPS180] and [FIPS202] **MUST** be supported for use with the variants of SLH-DSA as shown below:

```
id-slh-dsa-sha2-128s:  SHA-256
id-slh-dsa-sha2-128f:  SHA-256
id-slh-dsa-sha2-192s:  SHA-512
id-slh-dsa-sha2-192f:  SHA-512
id-slh-dsa-sha2-256s:  SHA-512
id-slh-dsa-sha2-256f:  SHA-512
id-slh-dsa-shake-128s: SHAKE128 with 256-bit output
id-slh-dsa-shake-128f: SHAKE128 with 256-bit output
id-slh-dsa-shake-192s: SHAKE256 with 512-bit output
id-slh-dsa-shake-192f: SHAKE256 with 512-bit output
id-slh-dsa-shake-256s: SHAKE256 with 512-bit output
id-slh-dsa-shake-256f: SHAKE256 with 512-bit output
```

The object identifiers for SHA-256 and SHA-512 are included in [RFC5754]. The object identifiers for SHAKE128 and SHAKE256 are included in [RFC8702]. In all four cases, the AlgorithmIdentifier **SHOULD NOT** include parameters.

signatureAlgorithm:
The signatureAlgorithm **MUST** contain one of the SLH-DSA algorithm identifiers, and the algorithm parameters field **MUST** be absent. The algorithm identifier **MUST** be one of the following:

```
id-slh-dsa-sha2-128s,  id-slh-dsa-sha2-128f,
id-slh-dsa-sha2-192s,  id-slh-dsa-sha2-192f,
id-slh-dsa-sha2-256s,  id-slh-dsa-sha2-256f,
id-slh-dsa-shake-128s, id-slh-dsa-shake-128f,
id-slh-dsa-shake-192s, id-slh-dsa-shake-192f,
id-slh-dsa-shake-256s, id-slh-dsa-shake-256f.
```

signature:
The signature contains the signature value resulting from the SLH-DSA signing operation with the parameters associated with the selected signatureAlgorithm. The SLH-DSA signature-generation operation is specified in Section 10.2.1 of [FIPS205], and the SLH-DSA signature verification operation is specified in Section 10.3 of [FIPS205]. Signature verification **MUST** include checking that the signatureAlgorithm field identifies SLH-DSA parameters that are consistent with public key used to validate the signature.

5. Security Considerations

Implementations **MUST** protect the private keys. Compromise of the private keys may result in the ability to forge signatures.

When generating an SLH-DSA key pair, an implementation **MUST** generate each key pair independently of all other key pairs in the SLH-DSA hypertree.

A SLH-DSA tree **MUST NOT** be used for more than 2^{64} signing operations.

The generation of private keys relies on random numbers. The use of inadequate Pseudorandom Number Generators (PRNGs) to generate these values can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute-force searching the whole key space. The generation of quality random numbers is difficult, and [RFC4086] offers important guidance in this area.

To avoid algorithm-substitution attacks, the CMSAlgorithmProtection attribute defined in [RFC6211] **SHOULD** be included in signed attributes.

Implementers should consider their particular use cases and may choose to implement optional fault-attack countermeasures [CMP2018] [Ge2023]. Verifying a signature before releasing the signature value is a typical fault-attack countermeasure; however, this countermeasure is not

effective for SLH-DSA [Ge2023]. Redundancy by replicating the signature-generation process **MAY** be used as an effective fault-attack countermeasure for SLH-DSA [Ge2023]; however, the SLH-DSA signature generation is already considered slow.

Likewise, implementers should consider their particular use cases and may choose to implement protections against passive power and emissions side-channel attacks [SLotH].

6. Operational Considerations

If slh_sign is implemented in a hardware device such as Hardware Security Module (HSM) or portable cryptographic token, implementations can avoid sending the full content to the device. By including signed attributes, which necessarily include the message-digest attribute and the content-type attribute as described in Section 5.3 of [RFC5652], the much smaller set of signed attributes are sent to the device for signing.

By including signed attributes in the SignerInfo, one can obtain similar interface characteristics to SLH-DSA in pre-hash mode. With pre-hash mode, the hash of the content is passed to the SLH-DSA signature operation instead of the full message content. By including signed attributes in the SignerInfo, a relatively small to-be-signed value is passed to the SLH-DSA signature operation. For this reason, SLH-DSA pre-hash mode is not specified for use with the CMS SignedData. Note SLH-DSA pre-hash mode always yields a different signature value than SLH-DSA pure mode, even if the to-be-signed content is the same.

When using SLH-DSA in pure mode, it is not possible to single-pass process the content to verify a SignedData message that does not contain signed attributes. To assist recipients that might make use of stream-based APIs, implementers **SHOULD** include signed attributes within any SignerInfo that uses SLH-DSA as signature algorithm. Doing so allows the recipient implementation to avoid keeping the signed content in memory. Recall that when signed attributes are present, they **MUST** contain a content-type attribute and a message-digest attribute, and they **SHOULD** contain a CMSAlgorithmProtection attribute.

7. IANA Considerations

For the ASN.1 Module in Appendix A, IANA has assigned an Object Identifier (OID) for the module identifier (81) with a Description of "id-mod-slh-dsa-2024". The OID for the module has been allocated in the "SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0)" registry.

8. References

8.1. Normative References

[FIPS180] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", NIST FIPS 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <https://doi.org/10.6028/NIST.FIPS.180-4>.

[FIPS202] National Institute of Standards and Technology (NIST), "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", NIST FIPS 202, DOI 10.6028/NIST.FIPS.202, August 2015, <https://doi.org/10.6028/NIST.FIPS.202>.

[FIPS205] National Institute of Standards and Technology (NIST), "Stateless Hash-Based Digital Signature Standard", NIST FIPS 205, DOI 10.6028/NIST.FIPS.205, 13 August 2024, <https://doi.org/10.6028/NIST.FIPS.205>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5754] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, DOI 10.17487/RFC5754, January 2010, <https://www.rfc-editor.org/info/rfc5754>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/info/rfc5958>.

[RFC6211] Schaad, J., "Cryptographic Message Syntax (CMS) Algorithm Identifier Protection Attribute", RFC 6211, DOI 10.17487/RFC6211, April 2011, <https://www.rfc-editor.org/info/rfc6211>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8702] Kampanakis, P. and Q. Dang, "Use of the SHAKE One-Way Hash Functions in the Cryptographic Message Syntax (CMS)", RFC 8702, DOI 10.17487/RFC8702, January 2020, <https://www.rfc-editor.org/info/rfc8702>.

[X680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

[X690] ITU-T, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1-2021, February 2021, <https://www.itu.int/rec/T-REC-X.690>.

8.2. Informative References

[CMP2018] Castelnovi, L., Martinelli, A., and T. Prest, "Grafting Trees: A Fault Attack Against the SPHINCS Framework", Post-Quantum Cryptography (PQCrypto 2018), Lecture Notes in Computer Science, vol. 10786, pp. 165-184, DOI 10.1007/978-3-319-79063-3_8, 2018, <https://link.springer.com/chapter/10.1007/978-3-319-79063-3_8>.

[Ge2023] Genêt, A., "On Protecting SPHINCS+ Against Fault Attacks", IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2023, no. 2, pp. 80-114, DOI 10.46586/tches.v2023.i2.80-114, 2023, <https://tches.iacr.org/index.php/TCHES/article/view/10278/9726>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC5911] Hoffman, P. and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, DOI 10.17487/RFC5911, June 2010, <https://www.rfc-editor.org/info/rfc5911>.

[RFC8391] Huelsing, A., Butin, D., Gazdag, S., Rijneveld, J., and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme", RFC 8391, DOI 10.17487/RFC8391, May 2018, <https://www.rfc-editor.org/info/rfc8391>.

[RFC8554] McGrew, D., Curcio, M., and S. Fluhrer, "Leighton-Micali Hash-Based Signatures", RFC 8554, DOI 10.17487/RFC8554, April 2019, <https://www.rfc-editor.org/info/rfc8554>.

[SLotH] Saarinen, M.-J., "Accelerating SLH-DSA by Two Orders of Magnitude with a Single Hash Unit", Cryptology ePrint Archive, Paper 2024/367, 2024, <https://eprint.iacr.org/2024/367.pdf>.

Appendix A. ASN.1 Module

This ASN.1 Module builds upon the conventions established in [RFC5911].

```
<CODE BEGINS>
SLH-DSA-Module-2024
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
    id-smime(16) id-mod(0) id-mod-slh-dsa-2024(81) }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
  PUBLIC-KEY, SIGNATURE-ALGORITHM, SMIME-CAPS
    FROM AlgorithmInformation-2009  -- in [RFC5911]
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-algorithmInformation-02(58) } ;
```

```
--
-- Object Identifiers
--

nistAlgorithms OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
  country(16) us(840) organization(1) gov(101) csor(3) 4 }

sigAlgs OBJECT IDENTIFIER ::= { nistAlgorithms 3 }

id-slh-dsa-sha2-128s OBJECT IDENTIFIER ::= { sigAlgs 20 }

id-slh-dsa-sha2-128f OBJECT IDENTIFIER ::= { sigAlgs 21 }

id-slh-dsa-sha2-192s OBJECT IDENTIFIER ::= { sigAlgs 22 }

id-slh-dsa-sha2-192f OBJECT IDENTIFIER ::= { sigAlgs 23 }

id-slh-dsa-sha2-256s OBJECT IDENTIFIER ::= { sigAlgs 24 }

id-slh-dsa-sha2-256f OBJECT IDENTIFIER ::= { sigAlgs 25 }

id-slh-dsa-shake-128s OBJECT IDENTIFIER ::= { sigAlgs 26 }

id-slh-dsa-shake-128f OBJECT IDENTIFIER ::= { sigAlgs 27 }

id-slh-dsa-shake-192s OBJECT IDENTIFIER ::= { sigAlgs 28 }

id-slh-dsa-shake-192f OBJECT IDENTIFIER ::= { sigAlgs 29 }

id-slh-dsa-shake-256s OBJECT IDENTIFIER ::= { sigAlgs 30 }

id-slh-dsa-shake-256f OBJECT IDENTIFIER ::= { sigAlgs 31 }

--
-- Signature Algorithm, Public Key, and Private Key
--

sa-slh-dsa-sha2-128s SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-sha2-128s
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-sha2-128s }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-sha2-128s } }

sa-slh-dsa-sha2-128f SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-sha2-128f
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-sha2-128f }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-sha2-128f } }

sa-slh-dsa-sha2-192s SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-sha2-192s
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-sha2-192s }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-sha2-192s } }

sa-slh-dsa-sha2-192f SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-sha2-192f
    PARAMS ARE absent
```

```
    PUBLIC-KEYS { pk-slh-dsa-sha2-192f }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-sha2-192f } }

sa-slh-dsa-sha2-256s SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-sha2-256s
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-sha2-256s }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-sha2-256s } }

sa-slh-dsa-sha2-256f SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-sha2-256f
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-sha2-256f }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-sha2-256f } }

sa-slh-dsa-shake-128s SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-shake-128s
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-shake-128s }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-shake-128s } }

sa-slh-dsa-shake-128f SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-shake-128f
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-shake-128f }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-shake-128f } }

sa-slh-dsa-shake-192s SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-shake-192s
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-shake-192s }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-shake-192s } }

sa-slh-dsa-shake-192f SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-shake-192f
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-shake-192f }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-shake-192f } }

sa-slh-dsa-shake-256s SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-shake-256s
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-shake-256s }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-shake-256s } }

sa-slh-dsa-shake-256f SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-slh-dsa-shake-256f
    PARAMS ARE absent
    PUBLIC-KEYS { pk-slh-dsa-shake-256f }
    SMIME-CAPS { IDENTIFIED BY id-slh-dsa-shake-256f } }

pk-slh-dsa-sha2-128s PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-sha2-128s
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }
```

```
pk-slh-dsa-sha2-128f PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-sha2-128f
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-sha2-192s PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-sha2-192s
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-sha2-192f PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-sha2-192f
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-sha2-256s PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-sha2-256s
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-sha2-256f PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-sha2-256f
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-shake-128s PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-shake-128s
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-shake-128f PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-shake-128f
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-shake-192s PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-shake-192s
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-shake-192f PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-shake-192f
```

```
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-shake-256s PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-shake-256s
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

pk-slh-dsa-shake-256f PUBLIC-KEY ::= {
    IDENTIFIER id-slh-dsa-shake-256f
    -- KEY no ASN.1 wrapping --
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    -- PRIVATE-KEY no ASN.1 wrapping -- }

SLH-DSA-PublicKey ::= OCTET STRING (SIZE (32 | 48 | 64))

SLH-DSA-PrivateKey ::= OCTET STRING (SIZE (64 | 96 | 128))

--
-- Expand the signature algorithm set used by CMS [RFC5911]
--

SignatureAlgorithmSet SIGNATURE-ALGORITHM ::=
    { sa-slh-dsa-sha2-128s |
      sa-slh-dsa-sha2-128f |
      sa-slh-dsa-sha2-192s |
      sa-slh-dsa-sha2-192f |
      sa-slh-dsa-sha2-256s |
      sa-slh-dsa-sha2-256f |
      sa-slh-dsa-shake-128s |
      sa-slh-dsa-shake-128f |
      sa-slh-dsa-shake-192s |
      sa-slh-dsa-shake-192f |
      sa-slh-dsa-shake-256s |
      sa-slh-dsa-shake-256f,
      ... }

--
-- Expand the S/MIME capabilities set used by CMS [RFC5911]
--

SMimeCaps SMIME-CAPS ::=
    { sa-slh-dsa-sha2-128s.&smimeCaps |
      sa-slh-dsa-sha2-128f.&smimeCaps |
      sa-slh-dsa-sha2-192s.&smimeCaps |
      sa-slh-dsa-sha2-192f.&smimeCaps |
      sa-slh-dsa-sha2-256s.&smimeCaps |
      sa-slh-dsa-sha2-256f.&smimeCaps |
      sa-slh-dsa-shake-128s.&smimeCaps |
      sa-slh-dsa-shake-128f.&smimeCaps |
      sa-slh-dsa-shake-192s.&smimeCaps |
      sa-slh-dsa-shake-192f.&smimeCaps |
      sa-slh-dsa-shake-256s.&smimeCaps |
```

```
      sa-slh-dsa-shake-256f.&smimeCaps,
      ... }

END

<CODE ENDS>
```

Acknowledgements

Thanks to Mike Ounsworth, Tomas Gustavsson, Daniel Van Geest, Carl Wallace, Phillip Hallam-Baker, Dieter Bratko, Vijay Gurbani, Paul Wouters, and Roman Danyliw for their careful review and constructive comments.

Authors' Addresses

Russ Housley
Vigil Security, LLC
Email: housley@vigilsec.com

Scott Fluhrer
Cisco Systems
Email: sfluhrer@cisco.com

Panos Kampanakis
Amazon Web Services
Email: kpanos@amazon.com

Bas Westerbaan
Cloudflare
Email: bas@westerbaan.name