

INTERNET-DRAFT
Internet Engineering Task Force (IETF)
Intended Status: Proposed Standard
Expires: 22 August 2019

R. Housley
Vigil Security

22 February 2019

Use of the HSS/LMS Hash-based Signature Algorithm
in the Cryptographic Message Syntax (CMS)
<[draft-ietf-lamps-cms-hash-sig-05](#)>

Abstract

This document specifies the conventions for using the the HSS/LMS hash-based signature algorithm with the Cryptographic Message Syntax (CMS). In addition, the algorithm identifier and public key syntax are provided. The HSS/LMS algorithm is one form of hash-based digital signature; it is described in [[HASHSIG](#)].

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

INTERNET-DRAFT

February 2019

Copyright and License Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	ASN.1	3
1.2.	Terminology	3
2.	HSS/LMS Hash-based Signature Algorithm Overview	3
2.1.	Hierarchical Signature System (HSS)	4
2.2.	Leighton-Micali Signature (LMS)	4
2.3.	Leighton-Micali One-time Signature Algorithm (LM-OTS)	5
3.	Algorithm Identifiers and Parameters	6
4.	HSS/LMS Public Key Identifier	7
5.	Signed-data Conventions	8
6.	Security Considerations	9
6.1.	Implementation Security Considerations	9
6.2.	Algorithm Security Considerations	9
7.	IANA Considerations	10
8.	Acknowledgements	11
9.	References	11
9.1.	Normative References	11
9.2.	Informative References	11
	Appendix: ASN.1 Module	13
	Author's Address	16

INTERNET-DRAFT

February 2019

[1.](#) Introduction

This document specifies the conventions for using the HSS/LMS hash-based signature algorithm with the Cryptographic Message Syntax (CMS) [\[CMS\]](#) signed-data content type. The Leighton-Micali Signature (LMS) system provides a one-time digital signature that is a variant of Merkle Tree Signatures (MTS). The Hierarchical Signature System (HSS) is built on top of the LMS system to efficiently scale for a larger numbers of signatures. The HSS/LMS algorithm is one form of hash-based digital signature, and it is described in [\[HASHSIG\]](#). The HSS/LMS signature algorithm can only be used for a fixed number of signing operations. The number of signing operations depends upon the size of the tree. The HSS/LMS signature algorithm uses small public keys, and it has low computational cost; however, the signatures are quite large. The HSS/LMS private key can be very small when the signer is willing to perform additional computation at signing time; alternatively, the private key can consume additional memory and provide a faster signing time.

Well, yes, there is quite a range of possible time/memory trade-offs available when storing the private key; if you need to, the private key can be expressed in quite a small amount of space (albeit at the expense of making the signature generation operation expensive).

[1.1.](#) ASN.1

CMS values are generated using ASN.1 [\[ASN1-B\]](#), using the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [\[ASN1-E\]](#).

[1.2.](#) 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.

[1.3.](#) Algorithm Considerations

At Black Hat USA 2013, some researchers gave a presentation on the current state of public key cryptography. They said: "Current cryptosystems depend on discrete logarithm and factoring which has seen some major new developments in the past 6 months" [[BH2013](#)]. They encouraged preparation for a day when RSA and DSA cannot be depended upon.

Housley

[Page 3]

INTERNET-DRAFT

February 2019

A post-quantum cryptosystem is a system that is secure against quantum computers that have more than a trivial number of quantum bits. It is open to conjecture when it will be feasible to build such a machine. RSA, DSA, and ECDSA are not post-quantum secure.

The LM-OTS one-time signature, LMS, and HSS do not depend on discrete logarithm or factoring, as a result these algorithms are considered to be post-quantum secure.

Hash-based signatures [[HASHSIG](#)] are currently defined to use exclusively SHA-256 [[SHS](#)]. An IANA registry is defined so that other hash functions could be used in the future. LM-OTS signature generation prepends a random string as well as other metadata before computing the hash value. The inclusion of the random value reduces the chances of an attacker being able to find collisions, even if the attacker has a large-scale quantum computer.

Today, RSA is often used to digitally sign software updates. This means that the distribution of software updates could be compromised if a significant advance is made in factoring or a quantum computer is invented. The use of HSS/LMS hash-based signatures to protect software update distribution, perhaps using the format described in [[FWPROT](#)], will allow the deployment of software that implements new cryptosystems.

[2.](#) HSS/LMS Hash-based Signature Algorithm Overview

Merkle Tree Signatures (MTS) are a method for signing a large but fixed number of messages. An MTS system depends on a one-time

signature method and a collision-resistant hash function.

This specification makes use of the hash-based algorithm specified in [\[HASHSIG\]](#), which is the Leighton and Micali adaptation [\[LM\]](#) of the original Lamport-Diffie-Winternitz-Merkle one-time signature system [\[M1979\]](#) [\[M1987\]](#) [\[M1989a\]](#) [\[M1989b\]](#).

As implied by the name, the hash-based signature algorithm depends on a collision-resistant hash function. The hash-based signature algorithm specified in [\[HASHSIG\]](#) currently uses only the SHA-256 one-way hash function [\[SHS\]](#), but it also establishes an IANA registry to permit the registration of additional one-way hash functions in the future.

[2.1](#). Hierarchical Signature System (HSS)

The MTS system specified in [\[HASHSIG\]](#) uses a hierarchy of trees. The Hierarchical N-time Signature System (HSS) allows subordinate trees to be generated when needed by the signer. Otherwise, generation of

the entire tree might take weeks or longer.

An HSS signature as specified in [\[HASHSIG\]](#) carries the number of signed public keys (*Nspk*), followed by that number of signed public keys, followed by the LMS signature as described in [Section 2.2](#). The public key for the top-most LMS tree is the public key of the HSS system. The LMS private key in the parent tree signs the LMS public key in the child tree, and the LMS private key in the bottom-most tree signs the actual message. The signature over the public key and the signature over the actual message are LMS signatures as described in [Section 2.2](#).

The elements of the HSS signature value for a stand-alone tree (a top tree with no children) can be summarized as:

```
u32str(0) ||  
lms_signature /* signature of message */
```

The elements of the HSS signature value for a tree with *Nspk* signed public keys can be summarized as:

```
u32str(Nspk) ||
```

```
signed_public_key[0] ||
signed_public_key[1] ||
...
signed_public_key[Nspk-2] ||
signed_public_key[Nspk-1] ||
lms_signature /* signature of message */
```

where, as defined in Section 3.3 of [\[HASHSIG\]](#), a signed_public_key is the lms_signature over the public key followed by the public key itself. Note that Nspk is the number of levels in the hierarchy of trees minus 1.

[2.2](#). Leighton-Micali Signature (LMS)

Each tree in the system specified in [\[HASHSIG\]](#) uses the Leighton-Micali Signature (LMS) system. LMS systems have two parameters. The first parameter is the height of the tree, h , which is the number of levels in the tree minus one. The [\[HASHSIG\]](#) specification supports five values for this parameter: $h=5$; $h=10$; $h=15$; $h=20$; and $h=25$. Note that there are 2^h leaves in the tree. The second parameter is the number of bytes output by the hash function, m , which is the amount of data associated with each node in the tree. The [\[HASHSIG\]](#) specification supports only the SHA-256 hash function [\[SHS\]](#), with $m=32$.

The [\[HASHSIG\]](#) specification supports five tree sizes:

```
LMS_SHA256_M32_H5;
LMS_SHA256_M32_H10;
LMS_SHA256_M32_H15;
LMS_SHA256_M32_H20; and
LMS_SHA256_M32_H25.
```

The [\[HASHSIG\]](#) specification establishes an IANA registry to permit the registration of additional tree sizes in the future.

The LMS public key is the string consists of four elements: the lms_algorithm_type from the list above, the otstype to identify the LM-OTS type as discussed in [Section 2.3](#), the private key identifier (I) as described in Section 5.3 of [\[HASHSIG\]](#), and the m -byte string

associated with the root node of the tree.

The LMS public key can be summarized as:

```
u32str(lms_algorithm_type) || u32str(otstype) || I || T[1]
```

An LMS signature consists of four elements: the number of the leaf (q) associated with the LM-OTS signature, an LM-OTS signature as described in [Section 2.3](#), a typecode indicating the particular LMS algorithm, and an array of values that is associated with the path through the tree from the leaf associated with the LM-OTS signature to the root. The array of values contains the siblings of the nodes on the path from the leaf to the root but does not contain the nodes on the path itself. The array for a tree with height h will have h values. The first value is the sibling of the leaf, the next value is the sibling of the parent of the leaf, and so on up the path to the root.

The four elements of the LMS signature value can be summarized as:

```
u32str(q) ||  
ots_signature ||  
u32str(type) ||  
path[0] || path[1] || ... || path[h-1]
```

[2.3](#). Leighton-Micali One-time Signature Algorithm (LM-OTS)

Merkle Tree Signatures (MTS) depend on a one-time signature method. [\[HASHSIG\]](#) specifies the use of the LM-OTS. An LM-OTS has five parameters.

- n - The number of bytes associated with the hash function. [\[HASHSIG\]](#) supports only SHA-256 [\[SHS\]](#), with $n=32$.

- H - A preimage-resistant hash function that accepts byte strings of any length, and returns an n -byte string.
- w - The width in bits of the Winternitz coefficients. [\[HASHSIG\]](#) supports four values for this parameter: $w=1$; $w=2$; $w=4$; and $w=8$.
- p - The number of n -byte string elements that make up the LM-OTS

signature.

ls – The number of left-shift bits used in the checksum function, which is defined in Section 4.4 of [\[HASHSIG\]](#).

The values of p and ls are dependent on the choices of the parameters n and w, as described in [Appendix B](#) of [\[HASHSIG\]](#).

The [\[HASHSIG\]](#) specification supports four LM-OTS variants:

```
LMOTS_SHA256_N32_W1;  
LMOTS_SHA256_N32_W2;  
LMOTS_SHA256_N32_W4; and  
LMOTS_SHA256_N32_W8.
```

The [\[HASHSIG\]](#) specification establishes an IANA registry to permit the registration of additional variants in the future.

Signing involves the generation of C, an n-byte random value.

The LM-OTS signature value can be summarized as the identifier of the LM-OTS variant, the random value, and a sequence of hash values that correspond to the elements of the public key as described in [Section 4.5](#) of [\[HASHSIG\]](#):

```
u32str(otstype) || C || y[0] || ... || y[p-1]
```

[3.](#) Algorithm Identifiers and Parameters

The algorithm identifier for an HSS/LMS hash-based signatures is:

```
id-alg-hss-lms-hashsig OBJECT IDENTIFIER ::= { iso(1)  
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)  
    smime(16) alg(3) 17 }
```

When this object identifier is used for a HSS/LMS signature, the AlgorithmIdentifier parameters field MUST be absent (that is, the parameters are not present; the parameters are not set to NULL).

The signature value is a large OCTET STRING. The signature format is

designed for easy parsing. Each format includes a counter and type

codes that indirectly providing all of the information that is needed to parse the value during signature validation.

The signature value identifies the hash function used in the HSS/LMS tree. In [\[HASHSIG\]](#) only the SHA-256 hash function [\[SHS\]](#) is supported, but it also establishes an IANA registry to permit the registration of additional hash functions in the future.

[4.](#) HSS/LMS Public Key Identifier

The AlgorithmIdentifier for an HSS/LMS public key uses the id-alg-hss-lms-hashsig object identifier, and the parameters field MUST be absent.

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; however, it MUST NOT contain other values.

```
pk-HSS-LMS-HashSig PUBLIC-KEY ::= {  
    IDENTIFIER id-alg-hss-lms-hashsig  
    KEY HSS-LMS-HashSig-PublicKey  
    PARAMS ARE absent  
    CERT-KEY-USAGE  
    { digitalSignature, nonRepudiation, keyCertSign, cRLSign } }
```

```
HSS-LMS-HashSig-PublicKey ::= OCTET STRING
```

Note that the id-alg-hss-lms-hashsig algorithm identifier is also referred to as id-alg-mts-hashsig. This synonym is based on the terminology used in an early draft of the document that became [\[HASHSIG\]](#).

The public key value is an OCTET STRING. Like the signature format, it is designed for easy parsing. The value is the number of levels in the public key, L, followed by the LMS public key.

The HSS/LMS public key value can be summarized as:

```
u32str(L) || lms_public_key
```

Note that the public key for the top-most LMS tree is the public key of the HSS system. When L=1, the HSS system is a single tree.

5. Signed-data Conventions

As specified in [\[CMS\]](#), the digital signature is produced from the message digest and the signer's private key. The signature is computed over different value depending on whether signed attributes are absent or present. When signed attributes are absent, the HSS/LMS signature is computed over the content. When signed attributes are present, a hash is computed over the content using the same hash function that is used in the HSS/LMS tree, and then a message-digest attribute is constructed with the resulting hash value, and then DER encode the set of signed attributes, which MUST include a content-type attribute and a message-digest attribute, and then the HSS/LMS signature is computed over the output of the DER-encode operation. In summary:

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

When using [\[HASHSIG\]](#), the fields in the SignerInfo are used as follows:

digestAlgorithm MUST contain the one-way hash function used to in the HSS/LMS tree. In [\[HASHSIG\]](#), SHA-256 is the only supported hash function, but other hash functions might be registered in the future. For convenience, the AlgorithmIdentifier for SHA-256 from [\[PKIXASN1\]](#) is repeated here:

```
mda-sha256 DIGEST-ALGORITHM ::= {
    IDENTIFIER id-sha256
    PARAMS TYPE NULL ARE preferredAbsent }
```

```
id-sha256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
    country(16) us(840) organization(1) gov(101) csor(3)
    nistAlgorithms(4) hashalgs(2) 1 }
```

signatureAlgorithm MUST contain id-alg-hss-lms-hashsig, and the algorithm parameters field MUST be absent.

signature contains the single HSS signature value resulting from the signing operation as specified in [\[HASHSIG\]](#).

6. Security Considerations

Implementations MUST protect the private keys. Compromise of the

private keys may result in the ability to forge signatures. Along with the private key, the implementation MUST keep track of which

leaf nodes in the tree have been used. Loss of integrity of this tracking data can cause an one-time key to be used more than once. As a result, when a private key and the tracking data are stored on non-volatile media or stored in a virtual machine environment, care must be taken to preserve confidentiality and integrity.

When generating a LMS key pair, an implementation MUST generate each key pair independently of all other key pairs in the HSS tree.

An implementation MUST ensure that a LM-OTS private key is used to generate a signature only one time, and ensure that it cannot be used for any other purpose.

The generation of private keys relies on random numbers. The use of inadequate pseudo-random 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.

The generation of hash-based signatures also depends on random numbers. While the consequences of an inadequate pseudo-random number generator (PRNGs) to generate these values is much less severe than the generation of private keys, the guidance in [[RFC4086](#)] remains important.

When computing signatures, the same hash function SHOULD be used to compute the message digest of the content and the signed attributes, if they are present.

[7.](#) IANA Considerations

SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0) registry, change the reference for value 64 to point to this document.

In the SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3)

registry, change the description for value 17 to "id-alg-hss-lms-hashsig" and change the reference to point to this document.

Also, add the following note to the registry:

Value 17, "id-alg-hss-lms-hashsig", is also referred to as "id-alg-mts-hashsig".

[8.](#) Acknowledgements

Many thanks to Scott Fluhrer, Jonathan Hammell, Panos Kampanakis, Jim Schaad, Sean Turner, and Daniel Van Geest for their careful review and comments.

[9.](#) References

[9.1.](#) Normative References

- [ASN1-B] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, 2015.
- [ASN1-E] 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, 2015.
- [CMS] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, [RFC 5652](#), DOI 10.17487/RFC5652, September 2009, <<http://www.rfc-editor.org/info/rfc5652>>.
- [HASHSIG] McGrew, D., M. Curcio, and S. Fluhrer, "Hash-Based Signatures", Work in progress. <[draft-mcgrew-hash-sigs-12](#)>
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<http://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>>.
- [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>>.
- [SHS] National Institute of Standards and Technology (NIST), FIPS Publication 180-3: Secure Hash Standard, October 2008.

Housley

[Page 11]

INTERNET-DRAFT

February 2019

[9.2.](#) Informative References

- [BH2013] Ptacek, T., T. Ritter, J. Samuel, and A. Stamos, "The Factoring Dead: Preparing for the Cryptopocalypse", August 2013. <<https://media.blackhat.com/us-13/us-13-Stamos-The-Factoring-Dead.pdf>>
- [CMSASN1] 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, <<http://www.rfc-editor.org/info/rfc5911>>.
- [CMSASN1U] Schaad, J. and S. Turner, "Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX)", [RFC 6268](#), DOI 10.17487/RFC6268, July 2011, <<http://www.rfc-editor.org/info/rfc6268>>.
- [FWPROT] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", [RFC 4108](#), DOI 10.17487/RFC4108, August 2005, <<http://www.rfc-editor.org/info/rfc4108>>.
- [LM] Leighton, T. and S. Micali, "Large provably fast and secure digital signature schemes from secure hash

functions", U.S. Patent 5,432,852, July 1995.

- [M1979] Merkle, R., "Secrecy, Authentication, and Public Key Systems", Stanford University Information Systems Laboratory Technical Report 1979-1, 1979.
- [M1987] Merkle, R., "A Digital Signature Based on a Conventional Encryption Function", Lecture Notes in Computer Science crypto87, 1988.
- [M1989a] Merkle, R., "A Certified Digital Signature", Lecture Notes in Computer Science crypto89, 1990.
- [M1989b] Merkle, R., "One Way Hash Functions and DES", Lecture Notes in Computer Science crypto89, 1990.
- [PKIXASN1] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", [RFC 5912](#), DOI 10.17487/RFC5912, June 2010, <<http://www.rfc-editor.org/info/rfc5912>>.

- [PQC] Bernstein, D., "Introduction to post-quantum cryptography", 2009.
<http://www.pqcrypto.org/www.springer.com/cda/content/document/cda_downloadaddocument/9783540887010-c1.pdf>
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", [BCP 106](#), [RFC 4086](#), DOI 10.17487/RFC4086, June 2005, <<http://www.rfc-editor.org/info/rfc4086>>.

Appendix: ASN.1 Module

MTS-HashSig-2013

```
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
  id-smime(16) id-mod(0) id-mod-mts-hashsig-2013(64) }
```

DEFINITIONS IMPLICIT TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS

PUBLIC-KEY, SIGNATURE-ALGORITHM, SMIME-CAPS

FROM AlgorithmInformation-2009 -- [RFC 5911](#) [[CMSASN1](#)]

{ iso(1) identified-organization(3) dod(6) internet(1)
security(5) mechanisms(5) pkix(7) id-mod(0)
id-mod-algorithmInformation-02(58) }

mda-sha256

FROM PKIX1-PSS-OAEP-Algorithms-2009 -- [RFC 5912](#) [[PKIXASN1](#)]

{ iso(1) identified-organization(3) dod(6)
internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
id-mod-pkix1-rsa-pkalgs-02(54) } ;

--

-- Object Identifiers

--

id-alg-hss-lms-hashsig OBJECT IDENTIFIER ::= { iso(1)
member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
smime(16) alg(3) 17 }

id-alg-mts-hashsig OBJECT IDENTIFIER ::= id-alg-hss-lms-hashsig

--

-- Signature Algorithm and Public Key

--

sa-HSS-LMS-HashSig SIGNATURE-ALGORITHM ::= {
IDENTIFIER id-alg-hss-lms-hashsig
PARAMS ARE absent
HASHES { mda-sha256 }
PUBLIC-KEYS { pk-HSS-LMS-HashSig }
SMIME-CAPS { IDENTIFIED BY id-alg-hss-lms-hashsig } }

```

pk-HSS-LMS-HashSig PUBLIC-KEY ::= {
    IDENTIFIER id-alg-hss-lms-hashsig
    KEY HSS-LMS-HashSig-PublicKey
    PARAMS ARE absent
    CERT-KEY-USAGE
        { digitalSignature, nonRepudiation, keyCertSign, cRLSign } }

HSS-LMS-HashSig-PublicKey ::= OCTET STRING

--
-- Expand the signature algorithm set used by CMS [CMSASN1U]
--

SignatureAlgorithmSet SIGNATURE-ALGORITHM ::=
    { sa-HSS-LMS-HashSig, ... }

--
-- Expand the S/MIME capabilities set used by CMS [CMSASN1]
--

SMimeCaps SMIME-CAPS ::=
    { sa-HSS-LMS-HashSig.&smimeCaps, ... }

END

```

Author's Address

Russ Housley
 Vigil Security, LLC
 516 Dranesville Road
 Herndon, VA 20170
 USA

EMail: housley@vigilsec.com