Use of the HSS/LMS Hash-based Signature Algorithm
in the Cryptographic Message Syntax (CMS)
<draft-ietf-lamps-cms-hash-sig-03>

Abstract

   This document specifies the conventions for using the the HSS/LMS
   hash-based signature algorithm with the Cryptographic Message Syntax
   (CMS). The HSS/LMS algorithm is one form of hash-based digital
   signature; it is described in [HASHSIG].

Status of this Memo

Table of Contents

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 HSS/LMS signature algorithm uses small
   private and public keys, and it has low computational cost; however,
   the signatures are quite large.

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.

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.
   Each signed public key is represented by the hash value at the root
   of the tree, and it also contains information about the tree
   structure. The signature over the public key is an LMS signature as
   described in Section 2.2.

   The elements of the HSS signature value for a stand-alone tree 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_on_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.

Currently, 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.

An LMS signature consists of four elements: the number of the leaf
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].

   Currently, 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:

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

3. Algorithm Identifiers and Parameters

   The algorithm identifier for an HSS/LMS hash-based signature when
   SHA-256 [SHS] is used to hash the content is the
   id-alg-hss-lms-hashsig-with-sha256 object identifier:

      id-alg-hss-lms-hashsig-with-sha256 OBJECT IDENTIFIER ::= { iso(1)
           member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
           smime(16) alg(3) TBD }

   The algorithm identifier for an HSS/LMS hash-based signature when
   SHA-384 [SHS] is used to hash the content is the
   id-alg-hss-lms-hashsig-with-sha384 object identifier:

      id-alg-hss-lms-hashsig-with-sha384 OBJECT IDENTIFIER ::= { iso(1)
           member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
           smime(16) alg(3) TBD }

   The algorithm identifier for an HSS/LMS hash-based signature when
   SHA-512 [SHS] is used to hash the content is the
   id-alg-hss-lms-hashsig-with-sha512 object identifier:

      id-alg-hss-lms-hashsig-with-sha512 OBJECT IDENTIFIER ::= { iso(1)
           member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
           smime(16) alg(3) TBD }

   When any of these object identifiers is used for a signature, the
   AlgorithmIdentifier parameters field MUST be absent (that is, the
   parameters are not present; the parameters are not set to NULL).

   The signature values 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.

4. HSS/LMS Public Key Identifier

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

   The SubjectPublicKeyInfo field of an X.509 certificate [RFC5280] is
   one place where this algorithm identifier appears. In this
   situation, 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
```

5. Signed-data Conventions

As specified in [CMS], the digital signature is produced from the
message digest and the signer's private key. If signed attributes
are absent, then the message digest is the hash of the content. If
signed attributes are present, then the hash of the content is placed
in the message-digest attribute, the set of signed attributes is DER
encoded, and the message digest is the hash of the encoded
attributes. In summary:

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

Sign(md)
```

When using [HASHSIG], the fields in the SignerInfo are used as
follows:

   digestAlgorithms SHOULD contain the one-way hash function used to
      compute the message digest on the eContent value. In
      [HASHSIG], SHA-256 is used throughout the hash tree, and the
      hash computation includes a random string. This random data
      makes it harder for an attacker to find collisions. The signer
      SHOULD use SHA-256 or a stronger hash function to compute the
      message digest on the content. For
       this purpose, Algorithm identifiers for SHA-256, SHA-384, and
      SHA-512 are provided in this document.

      Further, the same one-way hash function SHOULD be used to
      compute the message digest on both the eContent and the
      signedAttributes value if signedAttributes are present.

   signatureAlgorithm MUST contain id-alg-hss-lms-hashsig-with-
      sha256, id-alg-hss-lms-hashsig-with-sha384, or id-alg-hss-lms-
      hashsig-with-sha512. 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

6.1. Implementation 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 a LMS key pair is generating a LMS key pair, an implementation
   must must generate the key pair and the corresponding identifier
   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. RFC 4086 [RANDOM] 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.

6.2. Algorithm Security Considerations

   At Black Hat USA 2013, some researchers gave a presentation on the
   current sate 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.

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-OTP 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. An IANA registry is defined to 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.

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".

In the SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3)
registry, assign a new value for id-alg-hss-lms-hashsig-with-sha256
with a reference to this document.

In the SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3)
registry, assign a new value for id-alg-hss-lms-hashsig-with-sha384
with a reference to this document.

In the SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3)
registry, assign a new value for id-alg-hss-lms-hashsig-with-sha512
with a reference to this document.

8. Acknowledgements

Many thanks to 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>

[RFC2219]   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.

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_downloaddocument/9783540887010-c1.pdf>

   [RANDOM] 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, mda-sha384, mda-sha512
       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-hss-lms-hashsig-with-sha256 OBJECT IDENTIFIER ::= { iso(1)
         member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
         smime(16) alg(3) TBD }

    id-alg-hss-lms-hashsig-with-sha384 OBJECT IDENTIFIER ::= { iso(1)
         member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
         smime(16) alg(3) TBD }

    id-alg-hss-lms-hashsig-with-sha512 OBJECT IDENTIFIER ::= { iso(1)
         member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs9(9)
         smime(16) alg(3) TBD }

    --
    -- Signature Algorithm and Public Key
    --

    sa-HSS-LMS-HashSig-with-SHA256 SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-alg-hss-lms-hashsig-with-sha256
        PARAMS ARE absent
        HASHES { mda-sha256 }
        PUBLIC-KEYS { pk-HSS-LMS-HashSig }
        SMIME-CAPS { IDENTIFIED BY id-alg-hss-lms-hashsig-with-sha256 } }

    sa-HSS-LMS-HashSig-with-SHA384 SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-alg-hss-lms-hashsig-with-sha384
        PARAMS ARE absent
        HASHES { mda-sha384 }
        PUBLIC-KEYS { pk-HSS-LMS-HashSig }
        SMIME-CAPS { IDENTIFIED BY id-alg-hss-lms-hashsig-with-sha384 } }

    sa-HSS-LMS-HashSig-with-SHA512 SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-alg-hss-lms-hashsig-with-sha512
        PARAMS ARE absent
        HASHES { mda-sha512 }
        PUBLIC-KEYS { pk-HSS-LMS-HashSig }
        SMIME-CAPS { IDENTIFIED BY id-alg-hss-lms-hashsig-with-sha512 } }
```

```
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-with-SHA256 |
      sa-HSS-LMS-HashSig-with-SHA384 |
      sa-HSS-LMS-HashSig-with-SHA512, ... }

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

SMimeCaps SMIME-CAPS ::=
    { sa-HSS-LMS-HashSig-with-SHA256.&smimeCaps |
      sa-HSS-LMS-HashSig-with-SHA384.&smimeCaps |
      sa-HSS-LMS-HashSig-with-SHA512.&smimeCaps, ... }

END
```

Author's Address

    Russ Housley
    Vigil Security, LLC
    918 Spring Knoll Drive
    Herndon, VA 20170
    USA

    EMail: housley@vigilsec.com