INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: 24 October 2014

R. Housley
Vigil Security

24 April 2014

Use of the Hash-based Merkle Tree Signature (MTS) Algorithm in the Cryptographic Message Syntax (CMS) <draft-housley-cms-mts-hash-sig-01>

Abstract

This document specifies the conventions for using the Merkle Tree Signatures (MTS) digital signature algorithm with the Cryptographic Message Syntax (CMS). The MTS algorithm is one form of hash-based digital signature.

Copyright and License Notice

Table of Contents

1. Introduction

This document specifies the conventions for using the for using the Merkle Tree Signatures (MTS) digital signature algorithm with the Cryptographic Message Syntax (CMS) [CMS] signed-data content type. The MTS algorithm is one form of hash-based digital signature that can only be used for a specific number of signatures. The MTS algorithm is described in [HASHSIG]. The MTS algorithm uses small private and public keys, and it has low computational cost; however, the signatures are quite large.

CMS values are generated using ASN.1 [ASN1-02], using the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER).

1.1. MTS Digital Signature Algorithm

Merkle Tree Signatures (MTS) are a method for signing a large but fixed number of messages. An MTS system uses two cryptographic components: a one-time signature method and a collision-resistant hash function. Each MTS public/private key pair is associated with a k-way tree with each node containing an n-byte value. Each leaf of the tree contains the value of the public key of an Lamport, Diffie, Winternitz, and Merkle (LDWM) public/private key pair [HASHSIG]. The LDWM algorithm requires a robust one-way function to underpin the signature generation and verification. The algorithms in this document all make use of the SHA-256 [SHS] one-way hash function, which produces a 32 byte result.

The value at the root of the tree is the MTS public key. Each interior node is computed by applying the hash function to the concatenation of the values of its children nodes. Once again, the algorithms in this document all make use of the SHA-256 [SHS] one-way hash function.

An MTS signature consists of an LDWM signature, a node number that identifies the leaf node associated with the signature, and an array of values associated with the path through the tree from the LDWM signature leaf to the root. The array of values contains 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 branching number k and height h will have (k-1)*h values. The first (k-1) values are the siblings of the leaf, the next (k-1) values are the siblings of the parent of the leaf, and so on.

Four tree sizes are specified in [HASHSIG]:

MTS_SHA256_K2_H20:
- k = 2 (2 child nodes for each interior node),
- h = 20 (20 levels in the tree),
- n = 32 (32 bytes associated with each node), and
- mts_algorithm_type = 0x00000001.

MTS_SHA256_K4_H10:
- k = 4 (4 child nodes for each interior node),
- h = 10 (10 levels in the tree),
- n = 32 (32 bytes associated with each node), and
- mts_algorithm_type = 0x00000002.

MTS_SHA256_K8_H7:
- n = 8 (8 child nodes for each interior node),
- h = 7 (7 levels in the tree), and
- n = 32 (32 bytes associated with each node), and
- mts_algorithm_type = 0x00000003.

MTS_SHA256_K16_H5:
- k = 16 (16 child nodes for each interior node),
- h = 5 (5 levels in the tree),
- n = 32 (32 bytes associated with each node), and
- mts_algorithm_type = 0x00000004.

There are k^h leaves in the tree.

1.2. LDWM One-time Signature Algorithm

Merkle Tree Signatures (MTS) depend on a LDWM one-time signature method. The four variants described in [HASHSIG] depend on SHA-256 [SHS] and SHA-256-20, which is the same as SHA-256, except that the hash result is truncated to 20 bytes.

Four LDWN one-time signature algorithms are defined in [HASHSIG]:

LDWM_SHA256_M20_W1:
- ldwm_algorithm_type = 0x00000001; and
- the signature value is the 4-byte ldwm_algorithm_type followed by 265 20-byte values.

LDWM_SHA256_M20_W2:
- ldwm_algorithm_type = 0x00000002; and
- the signature value is the 4-byte ldwm_algorithm_type followed by 133 20-byte values.

LDWM_SHA256_M20_W4:
- ldwm_algorithm_type = 0x00000003; and
- the signature value is the 4-byte ldwm_algorithm_type followed by 67 20-byte values.

LDWM_SHA256_M20_W8:
- ldwm_algorithm_type = 0x00000004; and
- the signature value is the 4-byte ldwm_algorithm_type followed by 32 20-byte values.

1.3. Terminology

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 RFC 2119 [KEYWORDS].

2. Algorithm Identifiers and Parameters

The algorithm identifier for an MTS signature is id-alg-mts-hashsig:

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

id-alg  OBJECT IDENTIFIER ::= { id-smime  3 }

id-alg-mts-hashsig  OBJECT IDENTIFIER ::= { id-alg 17 }
```

When the id-alg-mts-hashsig algorithm identifier is used for a signature, the AlgorithmIdentifier parameters field MUST be absent.

The first 4 bytes of the signature value contains the mts_algorithm_type as defined in Section 4.5 of [HASHSIG]. For convenience, these values are repeated in above in Section 1.1 of this document. This value tells how to parse the remaining parts of the signature value, which is composed of an LDWM signature value, a 4-byte signature leaf number, and the MTS path.

The first 4 bytes of the LDWM signature value contains the ldwm_algorithm_type as defined in Section 3.10 of [HASHSIG]. For convenience, these values are repeated in above in Section 1.2 of this document.

The signature format is designed for easy parsing. Each format starts with a 4-byte enumeration value that indicates all of the details of the signature algorithm, indirectly providing all of the information that is needed to parse the value during signature validation.

3. Signed-data Conventions

digestAlgorithms SHOULD contain the one-way hash function used to compute the message digest on the eContent value. Since the hash-based signature algorithms all depend on SHA-256, it is strongly RECOMMENDED that SHA-256 also be used to compute the message digest on the content.

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 exist. Again, since the hash-based signature algorithms all depend on SHA-256, it is strongly RECOMMENDED that SHA-256 be used.

signatureAlgorithm MUST contain id-alg-mts-hashsig. The algorithm parameters field MUST be absent.

signature contains the single value resulting from the signing operation.

4. Security Considerations

4.1. Implementation Security Considerations

Implementations must protect the private keys. Compromise of the private keys may result in the ability to forge signatures. Further, a LDWM private key MUST be used only one time, and the LDWM private key MUST NOT 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.

When computing signatures, the same hash function SHOULD be used for all operations. This reduces the number of failure points in the signature process.

4.2. 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 whether it is feasible to build such a machine. RSA, DSA, and ECDSA are not post-quantum secure.

The LDWM one-time signature and MTS system do not depend on discrete logarithm or factoring, and these algorithms are considered to be post-quantum secure.

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 MTS signatures to protect software update distribution, perhaps using the format described in [FWPROT], will allow the deployment of software that implements new cryptosystems.

5. IANA Considerations

{{ RFC Editor: Please remove this section prior to publication. }}

This document has no actions for IANA.

6. Normative References

[ASN1-02] ITU-T, "ITU-T Recommendation X.680, X.681, X.682, and X.683", ITU-T X.680, X.681, X.682, and X.683, 2002.

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[HASHSIG] McGrew, D., and M. Curcio, "Hash-Based Signatures", Work in progress. <draft-mcgrew-hash-sigs-01>

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[SHS] National Institute of Standards and Technology (NIST), FIPS Publication 180-3: Secure Hash Standard, October 2008.

7. 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, June 2010.

[FWPROT] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, August 2005.

[PKIXASN1] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, June 2010.

[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, June 2005.

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 EXPLICIT TAGS ::= BEGIN

EXPORTS ALL;
```

```
   IMPORTS
     SIGNATURE-ALGORITHM PUBLIC-KEY
       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-smime  OBJECT IDENTIFIER ::= { iso(1) member-body(2)
               us(840) rsadsi(113549) pkcs(1) pkcs9(9) 16 }

   id-alg  OBJECT IDENTIFIER ::= { id-smime  3 }

   id-alg-mts-hashsig  OBJECT IDENTIFIER ::= { id-alg 17 }


   --
   -- Signature Algorithm and Public Key
   --

   sa-MTS-HashSig SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-alg-mts-hashsig
        HASHES { mda-sha256, ... }
        PUBLIC-KEYS { pk-MTS-HashSig } }

   pk-MTS-HashSig PUBLIC-KEY ::= {
       IDENTIFIER id-alg-mts-hashsig
       KEY MTS-HashSig-PublicKey }

   MTS-HashSig-PublicKey ::= OCTET STRING

   HashSignatureAlgs SIGNATURE-ALGORITHM ::= {
       sa-MTS-HashSig, ... }

   END
```

Author's Address

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

EMail: housley@vigilsec.com