WG Working Group
Internet-Draft
Intended status: Informational
Expires: 12 May 2023

S. Fluhrer
Cisco Systems
S. Gazdag
genua GmbH
D. V. Geest
ISARA Corporation
S. Kousidis
BSI
8 November 2022

Algorithm Identifiers for Hash-based Signatures for Use in the Internet X.509 Public Key Infrastructure
draft-gazdag-x509-hash-sigs-00

Abstract

This document specifies algorithm identifiers and ASN.1 encoding formats for the Hash-Based Signature (HBS) schemes Hierarchical Signature System (HSS), eXtended Merkle Signature Scheme (XMSS), and XMSS^MT, a multi-tree variant of XMSS, as well as SPHINCS+, the latter being the only stateless scheme. This specification applies to the Internet X.509 Public Key infrastructure (PKI) when digital signatures are used to sign certificates and certificate revocation lists (CRLs).

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-gazdag-x509-hash-sigs/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:WG@example.com), which is archived at https://example.com/WG.

Source for this draft and an issue tracker can be found at https://github.com/fluppe2/x509-hash-sigs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 12 May 2023.

Table of Contents

1. Introduction

Hash-Based Signature (HBS) Schemes combine Merkle trees with One/Few Time Signatures (OTS/FTS) in order to provide digital signature schemes that remain secure even when quantum computers become available. There security is well understood and depends only on the security of the underlying hash function. As such they can serve as an important building block for quantum computer resistant information and communication technology.

The private key of HSS, XMSS and XMSS^MT is a finite collection of OTS keys, hence only a limited number of messages can be signed and the private key's state must be updated and persisted after signing

to prevent reuse of OTS keys. Due to thise statefulness of the private key and the limited number of signatures that can be created, these signature algorithms might not be appropriate for use in interactive protocols. While the right selection of algorithm parameters would allow a private key to sign a virtually unbounded number of messages (e.g. 2^60), this is at the cost of a larger signature size and longer signing time. Since these algorithms are already known to be secure against quantum attacks, and because roots of trust are generally long-lived and can take longer to be deployed than end-entity certificates, these signature algorithms are more appropriate to be used in root and subordinate CA certificates. They are also appropriate in non-interactive contexts such as code signing. In particular, there are multi-party IoT ecosystems where publicly trusted code signing certificates are useful.

The private key of SPHINCS+ is a finite but very large collection of FTS keys and hence stateless. This typically comes at the cost of larger signatures compared to the stateful HBS variants. Thus SPHINCS+ is suitable for more use-cases if the signature sizes fit the requirements.

2. Conventions and Definitions

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.

The parameter 'n' is the security parameter, given in bytes. In practice this is typically aligned to the standard output length of the hash function in use, either 32 or 64 bytes. The height of a single tree is typically given by the parameter 'h'. The number of levels of trees is either called 'L' (HSS) or 'd' (XMSS^MT, SPHINCS+).

3. Subject Public Key Algorithms

Certificates conforming to [RFC5280] can convey a public key for any public key algorithm. The certificate indicates the algorithm through an algorithm identifier. An algorithm identifier consists of an OID and optional parameters.

In this document, we define new OIDs for identifying the different hash-based signature algorithms. An additional OID is defined in [RFC8708] and repeated here for convenience. For all of the OIDs, the parameters MUST be absent.

3.1. HSS Public Keys

The object identifier and public key algorithm identifier for HSS is defined in [RFC8708]. The definitions are repeated here for reference.

The object identifier for an HSS public key is "id-alg-hss-lms-hashsig":

```
  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 }
```

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 [RFC8554].

The HSS public key's properties are defined as follows:

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

The HSS public key is defined as follows:

```
  HSS-HashSig-PublicKey ::= SEQUENCE {
     levels     OCTET STRING, -- number of levels L
     tree       OCTET STRING, -- typecode of top-level LMS tree
     ots        OCTET STRING, -- typecode of top-level LM-OTS
     identifier OCTET STRING, -- identifier I of top-level LMS key pair
     root       OCTET STRING  -- root T[1] of top-level tree
  }
```

See [RFC8554] for more information on the contents and format of an HSS public key. Note that the single-tree signature scheme LMS is instantiated as HSS with level L=1.

3.2. XMSS Public Keys

The object identifier for an XMSS public key is id-alg-xmss-hashsig:

```
  id-alg-xmss-hashsig  OBJECT IDENTIFIER ::= { itu-t(0)
     identified-organization(4) etsi(0) reserved(127)
     etsi-identified-organization(0) isara(15) algorithms(1)
     asymmetric(1) xmss(13) 0 }
```

The XMSS public key's properties are defined as follows:

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

The XMSS public key is defined as follows:

```
  XMSS-HashSig-PublicKey ::= SEQUENCE {
     type       OCTET STRING, -- XMSS algorithm type
```

```
    seed       OCTET STRING, -- bitmask seed
    root       OCTET STRING  -- root of the single-tree
  }
```

See [RFC8391] for more information on the contents and format of an XMSS public key.

3.3. XMSS^MT Public Keys

The object identifier for an XMSS^MT public key is id-alg-xmssmt-hashsig:

```
  id-alg-xmssmt-hashsig  OBJECT IDENTIFIER ::= { itu-t(0)
     identified-organization(4) etsi(0) reserved(127)
     etsi-identified-organization(0) isara(15) algorithms(1)
     asymmetric(1) xmssmt(14) 0 }
```

The XMSS^MT public key's properties are defined as follows:

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

The XMSS^MT public key is defined as follows:

```
  XMSSMT-HashSig-PublicKey ::= SEQUENCE {
     type       OCTET STRING, -- XMSS^MT algorithm type
     seed       OCTET STRING, -- bitmask seed
     root       OCTET STRING  -- root of top-level tree
  }
```

See [RFC8391] for more information on the contents and format of an XMSS^MT public key.

3.4. SPHINCS+ Public Keys

The object identifier for a SPHINCS+ public key is id-alg-sphincsplus-hashsig:

```
  id-alg-sphincsplus-hashsig  OBJECT IDENTIFIER ::= { TBD }
```

The SPHINCS+ public key's properties are defined as follows:

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

  SPHINCSPLUS-HashSig-PublicKey ::= OCTET STRING
```

The SPHINCS+ public key is defined as follows:

```
XMSSMT-PublicKey ::= SEQUENCE {
   type       OCTET STRING, -- SPHINCS+ algorithm type
   seed       OCTET STRING, -- bitmask seed
   root       OCTET STRING  -- root of top-level tree
}
```

[SPHINCSPLUS] contains more information on the contents and format of a SPHINCS+ public key.

4. Key Usage Bits

The intended application for the key is indicated in the keyUsage certificate extension.

If the keyUsage extension is present in an end-entity certificate that indicates id-alg-xmss-hashsig or id-alg-xmssmt-hashsig in SubjectPublicKeyInfo, then the keyUsage extension MUST contain one or both of the following values:

nonRepudiation; and
digitalSignature.

If the keyUsage extension is present in a certification authority certificate that indicates id-alg-xmss-hashsig or id-alg-xmssmt-hashsig, then the keyUsage extension MUST contain one or more of the following values:

nonRepudiation;
digitalSignature;
keyCertSign; and
cRLSign.

[RFC8708] defines the key usage for id-alg-hss-lms-hashsig, which is the same as for the keys above.

5. Signature Algorithms

This section identifies OIDs for signing using HSS, XMSS, XMSS^MT, and SPHINCS+. When these algorithm identifiers appear in the algorithm field as an AlgorithmIdentifier, the encoding MUST omit the parameters field. That is, the AlgorithmIdentifier SHALL be a SEQUENCE of one component, one of the OIDs defined below.

The data to be signed is prepared for signing. For the algorithms used in this document, the data is signed directly by the signature algorithm, the data is not hashed before processing. Then, a private key operation is performed to generate the signature value. For HSS, the signature value is described in section 6.4 of [RFC8554]. For XMSS and XMSS^MT the signature values are described in sections B.2 and C.2 of [RFC8391], respectively. The octet string representing the signature is encoded directly in the BIT STRING without adding any additional ASN.1 wrapping. For the Certificate and CertificateList structures, the signature value is wrapped in the "signatureValue" BIT STRING field.

5.1. HSS Signature Algorithm

The HSS public key OID is also used to specify that an HSS signature was generated on the full message, i.e. the message was not hashed before being processed by the HSS signature algorithm.

```
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 }
```

The HSS signature is defined as follows:

```
HSS-HashSig-PublicKey ::= SEQUENCE {
   nspk            OCTET STRING, -- number of signed public keys
   signed_pub_keys OCTET STRING, -- nspk many LMS signed public keys
   signed_msg      OCTET STRING, -- an LMS signature of the message
}
```

Note that the number of signed public keys nspk equals L-1 where L denotes the number of levels. [RFC8391] contains more information on the contents and format of an HSS signature.

5.2. XMSS Signature Algorithm

The XMSS public key OID is also used to specify that an XMSS signature was generated on the full message, i.e. the message was not hashed before being processed by the XMSS signature algorithm.

```
id-alg-xmss-hashsig  OBJECT IDENTIFIER ::= { itu-t(0)
   identified-organization(4) etsi(0) reserved(127)
   etsi-identified-organization(0) isara(15) algorithms(1)
   asymmetric(1) xmss(13) 0 }
```

The XMSS signature is defined as follows:

```
XMSS-HashSig-Signature ::= SEQUENCE {
   index      OCTET STRING, -- index of the signature
   randomness OCTET STRING, -- a randomization string
   wots_sig   OCTET STRING, -- a WOTS+ signature
   auth_path  OCTET STRING, -- authentication path
}
```

auth_path consists of h (being the height of the tree) nodes.

The format of an XMSS signature is formally defined using XDR [RFC4506] and is defined in Appendix B.2 of [RFC8391].

5.3. XMSS^MT Signature Algorithm

The XMSS^MT public key OID is also used to specify that an XMSS^MT signature was generated on the full message, i.e. the message was not hashed before being processed by the XMSS^MT signature algorithm.

```
id-alg-xmssmt-hashsig  OBJECT IDENTIFIER ::= { itu-t(0)
   identified-organization(4) etsi(0) reserved(127)
   etsi-identified-organization(0) isara(15) algorithms(1)
```

```
    asymmetric(1) xmssmt(14) 0 }
```

The XMSS^MT signature is defined as follows:

```
  XMSSMT-HashSig-Signature ::= SEQUENCE {
     index      OCTET STRING, -- index of the signature
     randomness OCTET STRING, -- a randomization string
     xmss_sigs  OCTET STRING, -- d reduced XMSS signatures
  }
```

xmss_sigs consists of d (being the number levels) XMSS signatures in reduced form. Reduced form means that each XMSS signature contains only a WOTS+ signature and an authentication path, but no index and no randomization string.

The format of an XMSS^MT signature is is formally defined using XDR [RFC4506] and is defined in Appendix C.2 of [RFC8391].

5.4. SPHINCS+ Signature Algorithm

The SPHINCS+ public key OID is also used to specify that an SPHINCS+ signature was generated on the full message, i.e. the message was not hashed before being processed by the SPHINCS+ signature algorithm.

```
  id-alg-sphincsplus-hashsig  OBJECT IDENTIFIER ::= { TBD }
```

The SPHINCS+ signature is defined as follows:

```
  SPHINCSPLUS-HashSig-Signature ::= SEQUENCE {
     randomness OCTET STRING, -- a randomization string
     fors_sig   OCTET STRING, -- a FORS signature
     ht_sig     OCTET STRING, -- an HT signature
  }
```

fors_sig consists of k private key values and their associated authentication paths, while ht_sig consists of d (being the number of levels) XMSS signatures.

[SPHINCS] contains more information on the contents and format of a SPHINCS+ signature.

6. ASN.1 Module

For reference purposes, the ASN.1 syntax is presented as an ASN.1 module here.

```
-- ASN.1 Module

Hashsigs-pkix-0 -- TBD - IANA assigned module OID

DEFINITIONS EXPLICIT TAGS ::= BEGIN

IMPORTS PUBLIC-KEY, SIGNATURE-ALGORITHM FROM
AlgorithmInformation-2009 {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

-- -- id-alg-hss-lms-hashsig is defined in [ietf-lamps-cms-hash-sig]
-- -- 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-xmss-hashsig OBJECT IDENTIFIER ::= { itu-t(0) identified-
organization(4) etsi(0) reserved(127) etsi-identified-organization(0)
isara(15) algorithms(1) asymmetric(1) xmss(13) 0 }

id-alg-xmssmt-hashsig OBJECT IDENTIFIER ::= { itu-t(0) identified-
organization(4) etsi(0) reserved(127) etsi-identified-organization(0)
isara(15) algorithms(1) asymmetric(1) xmssmt(14) 0 }

id-alg-sphincsplus-hashsig OBJECT IDENTIFIER ::= { TBD }

-- Signature Algorithms and Public Keys

-- -- sa-HSS-LMS-HashSig is defined in [RFC8708] -- -- sa-HSS-LMS-
HashSig SIGNATURE-ALGORITHM ::= { -- IDENTIFIER id-alg-hss-lms-
hashsig -- PARAMS ARE absent -- PUBLIC-KEYS { pk-HSS-LMS-HashSig } --
SMIME-CAPS { IDENTIFIED BY id-alg-hss-lms-hashsig } }

-- -- pk-HSS-LMS-HashSig is defined in [RFC8708] -- -- 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

sa-XMSS SIGNATURE-ALGORITHM ::= { IDENTIFIER id-alg-xmss-hashsig
PARAMS ARE absent PUBLIC-KEYS { pk-XMSS } SMIME-CAPS { IDENTIFIED BY
id-alg-xmss-hashsig } }

pk-XMSS PUBLIC-KEY ::= { IDENTIFIER id-alg-xmss-hashsig KEY XMSS-
PublicKey PARAMS ARE absent CERT-KEY-USAGE { digitalSignature,
nonRepudiation, keyCertSign, cRLSign } }

XMSS-PublicKey ::= OCTET STRING

sa-XMSSMT SIGNATURE-ALGORITHM ::= { IDENTIFIER id-alg-xmssmt-hashsig
PARAMS ARE absent PUBLIC-KEYS { pk-XMSSMT } SMIME-CAPS { IDENTIFIED
BY id-alg-xmssmt-hashsig } }

pk-XMSSMT PUBLIC-KEY ::= { IDENTIFIER id-alg-xmssmt-hashsig KEY
XMSSMT-PublicKey PARAMS ARE absent CERT-KEY-USAGE { digitalSignature,
nonRepudiation, keyCertSign, cRLSign } }

XMSSMT-PublicKey ::= OCTET STRING

sa-SPHINCSPLUS SIGNATURE-ALGORITHM ::= { IDENTIFIER id-alg-
sphincsplus-hashsig PARAMS ARE absent PUBLIC-KEYS { pk-SPHINCSPLUS }
SMIME-CAPS { IDENTIFIED BY id-alg-sphincsplus-hashsig } }

pk-SPHINCSPLUS PUBLIC-KEY ::= { IDENTIFIER id-alg-sphincsplus-hashsig
```

```
KEY SPHINCSPLUS-PublicKey PARAMS ARE absent CERT-KEY-USAGE {
digitalSignature, nonRepudiation, keyCertSign, cRLSign } }

SPHINCSPLUS-PublicKey ::= OCTET STRING

END
```

7. Security Considerations

7.1. Algorithm Security Considerations

The cryptographic security of the signatures generated by the algorithms mentioned in this document depends only on the hash algorithms used within the signature algorithms and the pre-hash algorithm used to create an X.509 certificate's message digest. Grover's algorithm [Grover96] is a quantum search algorithm which gives a quadratic improvement in search time to brute-force pre-image attacks. The results of [BBBV97] show that this improvement is optimal, however [Fluhrer17] notes that Grover's algorithm doesn't parallelize well. Thus, given a bounded amount of time to perform the attack and using a conservative estimate of the performance of a real quantum computer, the pre-image quantum security of SHA-256 is closer to 190 bits. All parameter sets for the signature algorithms in this document currently use SHA-256 internally and thus have at least 128 bits of quantum pre-image resistance, or 190 bits using the security assumptions in [Fluhrer17].

[Zhandry15] shows that hash collisions can be found using an algorithm with a lower bound on the number of oracle queries on the order of 2^(n/3) on the number of bits, however [DJB09] demonstrates that the quantum memory requirements would be much greater. Therefore a parameter set using SHA-256 would have at least 128 bits of quantum collision-resistance as well as the pre-image resistance mentioned in the previous paragraph.

Given the quantum collision and pre-image resistance of SHA-256 estimated above, the current parameter sets used by id-alg-hss-lms-hashsig, id-alg-xmss-hashsig and id-alg-xmssmt-hashsig provide 128 bits or more of quantum security. This is believed to be secure enough to protect X.509 certificates for well beyond any reasonable certificate lifetime.

7.2. 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 a 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.

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. [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.

8. IANA Considerations

IANA is requested to assign a module OID from the "SMI for PKIX Module Identifier" registry for the ASN.1 module in Section 6.

9. References

9.1. Normative References

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

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

9.2. Informative References

[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://doi.org/10.17487/RFC8391>.

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

[RFC8708] Housley, R., "Use of the HSS/LMS Hash-Based Signature Algorithm in the Cryptographic Message Syntax (CMS)", RFC 8708, DOI 10.17487/RFC8708, February 2020, <https://doi.org/10.17487/RFC8708>.

Acknowledgments

Thanks for Russ Housley for the helpful suggestions.

This document uses a lot of text from similar documents ([RFC3279] and [RFC8410]) as well as [RFC8708]. Thanks go to the authors of those documents. "Copying always makes things easier and less error prone" - [RFC8411].

Authors' Addresses

Scott Fluhrer
Cisco Systems

Email: sfluhrer@cisco.com

S. Gazdag
genua GmbH

Email: ietf@gazdag.de

D. Van Geest
ISARA Corporation

Email: daniel.vangeest@isara.com

S. Kousidis
BSI

Email: tbd@tbd.tbd