International Standard

ISO/IEC 14888-4

First edition
2024-06

Information security — Digital signatures with appendix —

Part 4: Stateful hash-based mechanisms

Sécurité de l'information — Signatures digitales avec appendice —

Partie 4: Mécanismes basés sur le hachage dynamique

Reference number
ISO/IEC 14888-4:2024(en)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Information security, cybersecurity and privacy protection*.

A list of all parts in the ISO/IEC 14888 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Digital signatures with appendix are designed to offer integrity, authentication and non-repudiation. ISO/IEC 14888-2 specifies the class of digital signature mechanisms in which the security is based on the difficulty of integer factorization. ISO/IEC 14888-3 specifies the class in which the security is based on computing discrete logarithms. Unfortunately, if and when a large-scale general purpose quantum computer becomes available, all of these techniques will no longer be secure for practical key sizes.[1]

This document specifies a class of digital signatures whose security depends only on the security of the underlying hash function. At the time of publication of this document, standardized hash functions are believed to be secure even against attacks using large scale quantum computers. Hence, the schemes specified in this document do not suffer from the same problems as the schemes specified in ISO/IEC 14888-2 and ISO/IEC 14888-3.

The hash-based signature (HBS) schemes specified in this document are stateful schemes, whereby the private key is part of the state of the scheme. This means that at every signature generation state information held by the signer must be updated, as otherwise the security of the scheme is compromised. Therefore, when deploying any of the schemes specified in this document, it is expected that robust state-management practices are implemented to ensure that state information is correctly updated.

Information security — Digital signatures with appendix —

Part 4:
Stateful hash-based mechanisms

1 Scope

This document specifies stateful digital signature mechanisms with appendix, where the level of security is determined by the security properties of the underlying hash function.

This document also provides requirements for implementing basic state management, which is needed for the secure deployment of the stateful schemes described in this document.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
authentication path
list of hash values that show that a specific node belongs to a *Merkle tree* (3.5)

3.2
balanced binary tree
ordered tree in which each node has exactly two other nodes that are directly subordinate

3.3
binary tree
ordered tree in which each node has at most two other nodes that are directly subordinate

[SOURCE: ISO/IEC-2382:2015, 2121636, modified — notes to entry have been removed.]

3.4
L-tree
unbalanced *binary tree* (3.3) used to compress the Winternitz+ One-Time Signature Scheme (WOTS+) public keys in the eXtended Merkle Signature Scheme (XMSS)

3.5
Merkle tree
balanced binary tree (3.2), where each node of the tree corresponds to the hash of the labels of the child nodes

3.6
one-time signature
digital signature scheme where the security is limited to signing a single message for a given key pair

3.7
state
state information
information regarding key usage stored with an eXtended Merkle Signature Scheme (XMSS) or Leighton-Micali signature scheme (LMS) private key

3.8
state management
processes by which the *state* (3.7) is updated with each signature

Note 1 to entry: This is a crucial part of any stateful hash-based signature, as incorrect state information can lead to signature forgeries.

3.9
Winternitz parameter
parameter in the Winternitz *one-time signature* (3.6) scheme, which allows a trade-off between signature size and computation time

4 Symbols and abbreviated terms

4.1 Symbols

`0x00`	the hexadecimal representation of the 0 byte
$a \| b$	concatenation of the bit sequences a and b in the order specified
$a * b$	multiplication of a and b
a & b	bitwise logical AND of two bit strings a and b
a XOR b	bitwise logical EXCLUSIVE OR of two bit strings a and b
a mod b	the remainder when a is divided by b
$a \ll x$	the result of left-shifting a bit-string a by x positions (e.g. `0x08` << 1 = `0x10`)
$a \gg x$	the result of right-shifting a bit-string a by x positions (e.g. `0x08` >> 1 = `0x04`)
ceil(x)	the least integer greater than or equal to x
d	the number of layers of sub-trees in XMSS-MT
floor(x)	the greatest integer less than or equal to x
h	total height of the Merkle tree (or hyper tree for XMSS-MT and HSS) NOTE 1 h is the parameter which controls the maximum number of signatures, which are given by 2^h.
lb(x)	the base 2 logarithm of x
n	the length in bytes of one element of the private key, public key or signature, and the length of the message representative
m	the arbitrary length message to be signed in XMSS, XMSS-MT, LM-OTS, LMS, and HSS
M'	the message representative signed in XMSS and XMSS-MT
toByte(x,y)	computation of the y-byte string containing the binary representation of x in big-endian byte order

w	the Winternitz parameter used in the XMSS and XMSS-MT algorithms NOTE 2 w controls the trade-off between computation time and signature size. Smaller values of w/W lead to faster computations, while larger values of w lead to smaller signatures.
W	the Winternitz parameter used in the LMS and HSS algorithms. W and w are related as follows: $w = 2^W$.
$x[i]$	the ith element of the array x
$[X]_y$	the result of truncating X to its leftmost y bits, e.g. $[\texttt{0xfe}]_4 = \texttt{0xf}$

4.2 Abbreviated terms

HBS	hash-based signature
HSS	hierarchical signature scheme
LMS	Leighton-Micali signature scheme
OTS	one-time signature
RBG	random bit generator
WOTS+	Winternitz+ One-Time signature scheme
XMSS	eXtended Merkle Signature Scheme
XMSS-MT	eXtended Merkle Signature Scheme Multi Tree

5 XMSS and XMSS-MT

5.1 General

The XMSS scheme is a stateful hash-based signature scheme for which only a limited number of signatures can be created using a particular private key. The security of XMSS is based on the hardness of the Target Collision Resistance (TCR) problem, and XMSS has been proven to be secure in the standard model (see Reference [13]).

The XMSS-MT scheme, a variant of the XMSS scheme, is a stateful hash-based signature scheme that supports a larger number of signatures than XMSS. XMSS-MT inherits all the security properties of XMSS.[14]

The OIDs for these algorithms shall be in accordance with Annex A. Test vectors can be found in Annex C.

5.2 Common building blocks

5.2.1 General

The XMSS and the XMSS-MT schemes are described in a unified way since they employ common building blocks.

5.2.2 Address format

The *ADRS* input has three different address formats (see Table 1). Each layout is appropriate for a different step of the algorithms:

— the OTS address format is for the hash calls in the one-time signature schemes;

— the L-tree address format is for hashes used in the L-trees;

— the hash tree address format is for the Merkle-tree construction.

An L-tree is an unbalanced binary hash tree used to compute the leaves of the main XMSS binary hash tree.

Each *ADRS* address format consists of six 32-bit fields and one 64-bit field. The fields are encoded as unsigned integers. The *layerAddress* field refers to the layer in a multi-tree construction (0 if not using multi-trees). The *treeAddress* refers to the position (from left to right) of a tree in a multi-tree construction (0 if not using multi-trees). The *field* type differentiates the three *ADRS* layouts: 0 for OTS, 1 for L-tree and 2 for Hash tree. The field *keyAndMask* is used to distinguish if the hash call intends to generate a key (0) or a bitmask (1) for an OTS address. In the case of the L-tree and Hash tree address, *keyAndMask* is: (0) to generate a key, (1) to generate the most significant n bytes of the $2n$-byte bitmask and (2) to generate the least significant n bytes of the $2n$-byte bitmask.

Table 1 — XMSS address formats

OTS address	L-tree address	Hash tree address
layerAddress (32 bits)	layerAddress (32 bits)	layerAddress (32 bits)
treeAddress (64 bits)	treeAddress (64 bits)	treeAddress (64 bits)
type = 0 (32 bits)	type = 1 (32 bits)	type = 2 (32 bits)
OTSAddress (32 bits)	ltreeAddress (32 bits)	padding = 0 (32 bits)
chainAddress (32 bits)	treeHeight (32 bits)	treeHeight (32 bits)
hashAddress (32 bits)	treeIndex (32 bits)	treeIndex (32 bits)
keyAndMask (32 bits)	keyAndMask (32 bits)	keyAndMask (32 bits)
NOTE Each XMSS Address in this table is 32-bytes.		

The field *OTSAddress* encodes the index of the OTS key pair within the tree, the *chainAddress* encodes the chain index and finally the *hashAddress* encodes the hash function call index within the chain.

In the L-tree address layout, the *ltreeAddress* field encodes the index of the leaf computed with this L-tree. The *treeHeight* encodes the height of the node used as input for the next computation inside the L-tree. The *treeIndex* encodes the index of the node at that height, inside the L-tree.

In the tree hash address format, the *padding* field is always 0. The *treeHeight* encodes the height of the tree node being used for the next computation, followed by the *treeIndex* which is the node index at that height. For the L-tree address layout, the *keyAndMask* field can be: 0 (key), 1 (first bitmask), and 2 (second bitmask).

5.2.3 Required cryptographic functions

5.2.3.1 General

The XMSS scheme uses the following cryptographic primitives:

— Keyed Hash-Function F(*KEY*, *M*), where *KEY* is an n-byte key, *M* is an n-byte message and the output is n bytes.

— Keyed Hash-Function H(*KEY*, *M*), where *KEY* is an n-byte key, *M* is a $2n$-byte message and the output is n bytes.

— Keyed Hash-Function H_msg(*KEY*, *M*), where *KEY* is a $3n$-byte key, *M* is an arbitrary length message and the output is n bytes.

— Pseudo-Random Function PRF(*KEY*, *M*), where *KEY* is an n-byte key, *M* is a 32-byte message and the output is n bytes.

— Pseudo-Random Function $PRF_{Keygen}(KEY, M)$, where *KEY* is an *n*-byte key, *M* is a 32-byte message and the output is *n* bytes. This function is optional and only used if WOTS+ private keys are generated pseudo-randomly.

— Pseudo-Random Function $PRF_{Keygen\_MT}(KEY, M)$, where *KEY* is an *n*-byte key, *M* is a 32-byte message and the output is *n* bytes. This function is only used in XMSS_keygen if called from XMSS_MT_keygen.

— A non-deterministic random number generator RBG that provides at least a security level of 8*n* bits.

These functions shall be implemented with SHA2-256 (Dedicated Hash-Function 4 defined in ISO/IEC 10118-3), SHAKE256 (see ISO/IEC 10118-3:2018, C.2) as described below (see Annex B for further considerations on these instantiations).

5.2.3.2 Functions Based on SHA2-256

When using SHA2-256 as the underlying hash function and *n*=32, the following constructions shall be used.

— F(*KEY*, *M*): SHA2-256(toByte(0, 32) || *KEY* || *M*)

— H(*KEY*, *M*): SHA2-256(toByte(1, 32) || *KEY* || *M*)

— H_msg(*KEY*, *M*): SHA2-256(toByte(2, 32) || *KEY* || *M*)

— PRF(*KEY*, *M*): SHA2-256(toByte(3, 32) || *KEY* || *M*)

— $PRF_{Keygen}(KEY, M)$: SHA2-256(toByte(4, 32) || *KEY* || *M*)

— $PRF_{Keygen\_MT}(KEY, M)$: SHA2-256(toByte(5, 32) || *KEY* || *M*)

When using SHA2-256 as the underlying hash function and *n*=24, the following constructions shall be used.

— F(*KEY*, *M*): SHA2-256/192(toByte(0, 4) || *KEY* || *M*)

— H(*KEY*, *M*): SHA2-256/192(toByte(1, 4) || *KEY* || *M*)

— H_msg(*KEY*, *M*): SHA2-256/192(toByte(2, 4) || *KEY* || *M*)

— PRF(*KEY*, *M*): SHA2-256/192(toByte(3, 4) || *KEY* || *M*)

— $PRF_{Keygen}(KEY, M)$: SHA2-256/192(toByte(4, 4) || *KEY* || *M*)

— $PRF_{Keygen\_MT}(KEY, M)$: SHA2-256/192(toByte(5, 4) || *KEY* || *M*)

5.2.3.3 Functions Based on SHAKE

When using SHAKE256 and *n*=32, the following constructions shall be used.

— F(*KEY*, *M*): SHAKE256(toByte(0, 32) || *KEY* || *M*, 256)

— H(*KEY*, *M*): SHAKE256(toByte(1, 32) || *KEY* || *M*, 256)

— H_msg(*KEY*, *M*): SHAKE256(toByte(2, 32) || *KEY* || *M*, 256)

— PRF(*KEY*, *M*): SHAKE256(toByte(3, 32) || *KEY* || *M*, 256)

— $PRF_{Keygen}(KEY, M)$: SHAKE256(toByte(4, 32) || *KEY* || *M*, 256)

— PRF_{Keygen_MT}(KEY, M): SHAKE256(toByte(5, 32) || *KEY* || *M*, 256)

When using SHAKE256 and *n*=24, the following constructions shall be used.

— F(*KEY*, *M*): SHAKE256(toByte(0, 4) || *KEY* || *M*, 192)

— H(*KEY*, *M*): SHAKE256(toByte(1, 4) || *KEY* || *M*, 192)

— H_msg(*KEY*, *M*): SHAKE256(toByte(2, 4) || *KEY* || *M*, 192)

— PRF(*KEY*, *M*): SHAKE256(toByte(3, 4) || *KEY* || *M*, 192)

— PRF_{Keygen}(*KEY*, *M*): SHAKE256(toByte(4, 4) || *KEY* || *M*, 192)

— PRF_{Keygen_MT}(*KEY*, *M*): SHAKE256(toByte(5, 4) || *KEY* || *M*, 192)

5.2.4 Auxiliary functions

5.2.4.1 General

Both XMSS and XMSS-MT make use of two auxiliary functions, namely the base_w function and the chain function. The specifications of these auxiliary functions are given in 5.2.4.2 and 5.2.4.3.

5.2.4.2 base_w auxiliary function

This function is used in the signing and verification process to convert a string of bytes into a sequence of base *w* integers (i.e. integers between 0 and *w*-1).

Algorithm: base_w(*X*, *w*, *out_len*).

Input: A sequence of bytes *X* = *X*[0], … , *X*[*len_x* - 1] of length *len_x*, the base *w* which shall be 4 or 16, and the output length *out_len*.

Output: A sequence of integers *Q* = *Q*[0],…, *Q*[*out_len*-1] of length *out_len* from the set {0,1,…,*w*-1}

Steps:

a) Set *in* = 0, *out* = 0, *total* = 0, *bits* = 0.

b) For *i* from 0 to *out_len* - 1:

   1) If *bits* is equal to 0 then

      i) Set *total* = *X*[*in*].

      ii) Set *in* = *in* + 1.

      iii) Set *bits* = 8.

   2) Set *bits* = *bits* - lb(*w*).

   3) Set *Q*[*out*] = (*total* >> *bits*) & (*w* - 1).

   4) Set *out* = *out* + 1.

c) Return *Q*.

5.2.4.3 Chain auxiliary function

The chain function is the main building block of the XMSS and XMSS-MT schemes. This is the function used multiple times to produce the public key and signature from the private key material.

Algorithm: chain(*X*, *i*, *s*, *SEED*, *ADRS*).

Input: A string *X*, starting index *i*, length *s* of the hash chain to be computed, *n*-byte value *SEED*, 32-byte value *ADRS* formatted as an OTS address according to Table 1.

Output: An *n*-byte value output

Steps:

a) If s is equal to 0, then return X.

b) If $(i + s) > w - 1$, then return *NULL*.

c) Set *tmp* = chain(*X, i, s - 1, SEED, ADRS*).

d) Set *ADRS.hashAddress* = (*i + s - 1*).

e) Set *ADRS.keyAndMask* = 0.

f) Set *KEY* = PRF(*SEED, ADRS*).

g) Set *ADRS.keyAndMask* = 1.

h) Set *BM* = PRF(*SEED*, *ADRS*).

i) Return F(*KEY*, *tmp* XOR *BM*).

5.2.5 WOTS+ One-Time Signature Auxiliary Scheme

5.2.5.1 General

Both XMSS and XMSS-MT make use of an underlying one-time signature scheme called WOTS+. A one-time signature scheme has limited applicability because a signing key can only be used to sign a single message. If the same one-time signing key is used twice, the scheme loses its security guarantees (in particular, signature forgery becomes possible). WOTS+ shall not be used outside the context of XMSS and XMSS-MT. 5.2.5.2 gives the definition of WOTS+ key generation, signing and verification, which are algorithms used by XMSS/XMSS-MT key generation, signing and verification, respectively.

5.2.5.2 WOTS+ key generation

5.2.5.2.1 General

5.2.5.2.2, 5.2.5.2.3 and 5.2.5.2.4 describe the key generation process for WOTS+. There are two possible methods to create the private key: 1) generate the private key from a single seed (5.2.5.2.2), and 2) generate a uniform random private key (5.2.5.2.3). The main difference is that the method described in 5.2.5.2.2 is more memory-efficient as it uses a pseudo-random function to generate all WOTS+ private key elements from a single seed, while the one described in 5.2.5.2.3 does not introduce an additional security assumption regarding the pseudo-random function step.

Throughout the document, various other algorithms are required to generate WOTS+ private keys. For those, it is always assumed that the pseudo-random method described in 5.2.5.2.2 is used. Trivial changes (omitted for simplicity) to these algorithms are allowed so that the method described in 5.2.5.2.3 is used instead, at the cost of storing all WOTS+ private keys. The public key generation given in 5.2.5.2.4 is the same for both private key generation methods.

5.2.5.2.2 WOTS+ Private Key (pseudo-random)

This algorithm generates a WOTS+ private key. This method is called by the XMSS and XMSS-MT signatures schemes and makes use of *SK_S*, which works as a seed for the generation of all WOTS+ private keys. This method also receives an index *idx* which specifies what WOTS+ private key is required to be generated (among the 2^h possible ones), and the layer L and tree address T to identify the XMSS sub-tree (relevant when using the XMSS-MT variant).

Algorithm: WOTS+_generate_privkey(*SK_S*, *SEED*, *idx*, *L*, *T*).

Input: An n-byte string *SK_S*, public seed *SEED*, an integer *idx*, layer *L*, tree address *T*.

Output: A sequence of n-byte values *sk* of length *len*.

Steps:

a) Set *ADRS* = toByte(0, 32).

b) Set *ADRS.layerAddress* = *L*.

c) Set *ADRS.treeAddress* = *T*.

d) Set *ADRS.OTSAddress* = *idx*.

e) Set *ADRS.hashAddress* = 0.

f) For *i* from 0 to *len* - 1:

   1) Set *ADRS.chainAddress* = *i*.

   2) Set *sk*[*i*] = $\mathrm{PRF}_{\mathrm{Keygen}}$(*SK_S*, *SEED* || *ADRS*)

g) Return *sk*.

5.2.5.2.3 WOTS+ private key (random)

This algorithm generates a WOTS+ private key.

Algorithm: WOTS+_generate_privkey_random().

Input: None.

Output: A sequence of *n*-byte values *sk* of length *len*.

Steps:

a) For *i* from 0 to *len* - 1:

   1) Let *sk*[*i*] be an *n*-byte string from the output generation function of an RBG

b) Return *sk*.

5.2.5.2.4 WOTS+ public key

This algorithm generates a WOTS+ public key from a private key. It should be noted that the *SEED* used here is public information that is also made available to the verifier.

Algorithm: WOTS+_generate_pubkey(*sk*, *SEED*, *ADRS*).

Input: Sequence of *n*-byte strings *sk* of length *len*, *n*-byte *SEED*, a 32-byte *ADRS* formatted as an OTS address according to Table 1.

Output: Sequence of *n*-byte strings *pk* of length *len*.

Steps:

a) For *i* from 0 to *len* - 1:

   1) Set *ADRS.chainAddress* = *i*.

   2) *Set pk*[*i*] = chain(*sk*[*i*], 0, *w* - 1, *SEED*, *ADRS*).

b) Return *pk*.

5.2.5.3 WOTS+ signing

This algorithm generates a signature from a private key and a message. The values len_1 and len_2 used in WOTS+_sign are constants derived from the parameters n and w:

— len_1 = ceil(8*n/lb(w))

— len_2 = floor(lb(len_1 * (w - 1)) / lb(w)) + 1

Algorithm: WOTS+_sign(*sk*, *m*, *SEED*, *ADRS*).

Input: A private key *SK*, a message *M'* of length *n* bytes to be signed, *n*-byte *SEED*, a 32-byte *ADRS* formatted as an OTS address according to Table 1.

Output: Signature *sig* for the message *M'*.

Steps:

a) Set *csum* = 0.

b) Set *msg* = base_w(*M'*, *w*, len_1).

c) For *i* from 0 to len_1 - 1:

   1) Set *csum* = *csum* + *w* - 1 - *msg*[*i*].

d) Set *csum* = *csum* << (8 - ((len_2 * lb(w)) mod 8)).

e) Set *len2_bytes* = ceil((len_2 * lb(w)) / 8).

f) Set *msg* = *msg* || base_w(toByte(*csum*, *len2_bytes*), *w*, len_2).

g) For *i* from 0 to *len* - 1:

   1) Set *ADRS.chainAddress* = *i*.

   2) Set *sig*[*i*] = chain(*SK*[*i*], 0, *msg*[*i*], *SEED*, *ADRS*).

h) Return *sig*.

5.2.5.4 WOTS+ verification

5.2.5.4.1 Compute the WOTS+ public key

This algorithm verifies if a signature is authentic given a signature, a public key and a message. The WOTS+ verification process is divided into two parts. The first one recomputes the WOTS+ public key from the WOTS+ signature. The second one checks if the recomputed WOTS+ public key matches the original one.

When used in the context of XMSS or XMSS-MT signature verification, only the first part of the WOTS+ verification process is performed, i.e. the WOTS+ public key recovery step. It is never compared with any other WOTS+ public key. This verification is implicitly done at the XMSS or XMSS-MT signature verification level.

This algorithm computes a WOTS+ public key from a message and its signature.

Algorithm: WOTS+_pk_from_sig(*sig*, *M'*, *SEED*, *ADRS*).

Input: A signature *sig*, a message *M'* of length *n* bytes, *n*-byte *SEED*, a 32-byte *ADRS* formatted as an OTS address according to Table 1.

Output: A public key candidate *tmp_pk*.

Steps:

a) Set *csum* = 0.

b) Set *msg* = base_w(*M'*, *w*, len_1).

c) For *i* from 0 to len_1 - 1:

   1) Set *csum* = *csum* + *w* - 1 - *msg*[*i*].

d) Set *csum* = *csum* << (8 - ((len_2 * lb(*w*)) mod 8)).

e) Set *len2_bytes* = ceil((len_2 * lb(*w*)) / 8).

f) Set *msg* = *msg* || base_w(toByte(*csum*, *len2_bytes*), *w*, len_2).

g) For *i* from 0 to len - 1:

   1) Set *ADRS.chainAddress* = *i*.

   2) Set *tmp_pk*[*i*] = chain(*sig*[*i*], *msg*[*i*], *w* - 1 - *msg*[*i*], *SEED*, *ADRS*).

h) Return *tmp_pk*.

5.3 XMSS Algorithms

5.3.1 General

The XMSS scheme is a multi-time stateful signature scheme that allows for a large but fixed number of signatures. It uses the WOTS+ one-time signature scheme as a building block. It also uses a perfectly balanced binary tree, called a Merkle tree. This binary tree has height *h*/*d* (for XMSS, *d* = *1* and therefore *h*/*d* = *h*, while for XMSS-MT *d* > *1*). For the sake of simplicity, the height can be considered to be *h*, since this refers to XMSS specifically. A tree of height *h* has exactly 2^h leaf nodes. Each leaf node represents one WOTS+ key pair that can be used to generate one signature. In practice, such a binary tree represents a set of 2^h WOTS+ key pairs, thus an XMSS instantiation with parameter *h* allows up to 2^h signatures.

To prevent reusing the same WOTS+ key pair, each leaf node (which is associated with a WOTS+ key pair) is numbered from 0 to (2^h - 1), where the leftmost leaf node has index 0. After producing a new WOTS+ signature, the index shall be increased by 1. This process shall be flawlessly performed since reusing the same WOTS+ key pair more than once voids its security guarantees. This process makes the XMSS scheme a stateful scheme, meaning that the signer shall flawlessly update this index after each signature.

An XMSS signature essentially consists of two things: a WOTS+ signature and the authentication path, which is a set of nodes of the Merkle tree. The WOTS+ signature is used to re-generate the WOTS+ public key associated with the current leaf node. The WOTS+ public key plus the authentication path are enough information for a verifier to reconstruct the root of the Merkle tree. If the re-computed root node matches with the XMSS public key, the verifier then knows that the signature is authentic.

There are several methods to compute authentication paths, and each one offers a different performance profile, although all of them should compute exactly the same authentication path for a given signature.

The key generation and signing algorithms (for both XMSS and XMSS-MT) assume that the pseudo-random WOTS+ private key generation method described in 5.2.5.2.2 is used. In this case, the WOTS+ private keys are generated from a seed *SK_S* contained in the private key *SK*. If the random method described in 5.2.5.2.3 is used instead, the XMSS private key will not have a *SK_S* element, but rather the WOTS+ private keys instead.

5.3.2 Auxiliary functions

5.3.2.1 RAND_HASH

This algorithm implements the randomized tree hashing.

Algorithm: RAND_HASH(*LEFT*, *RIGHT*, *SEED*, *ADRS*).

Input: An n-byte string *LEFT*, an n-byte string *RIGHT*, an n-byte string *SEED*, a 32-byte string *ADRS* formatted according to Table 1 as either an L-tree Address or a Hash tree Address.

Output: An n-byte string.

Steps:

a) Set *ADRS.keyAndMask* = 0.

b) Set *KEY* = PRF(*SEED*, *ADRS*).

c) Set *ADRS.keyAndMask* = 1.

d) Set BM_0=PRF(*SEED*, *ADRS*).

e) Set *ADRS.keyAndMask* = 2.

f) Set BM_1 = PRF(*SEED*, *ADRS*).

g) Return H(*KEY*, (*LEFT* XOR BM_0) || (*RIGHT* XOR BM_1)).

5.3.2.2 L_TREE

This algorithm compresses a WOTS+ public key *pk* into a single n-byte value *pk*[0].

Algorithm: L_TREE(*pk, SEED, ADRS*).

Input: A sequence of n-byte strings *pk* of length *len*, public *SEED*, and 32-byte address *ADRS* formatted according to Table 1 as an L-tree Address. It should be noted that this method modifies the input *pk*.

Output: An n-byte string *pk*[0].

Steps:

a) Set *len′* = *len*.

b) Set *ADRS.treeHeight* = 0.

c) While *len′* > 1 do:

   1) For *i* from 0 to floor(*len′*/ 2) - 1:

      i) Set *ADRS.treeIndex* = *i*.

      ii) Set *pk*[*i*] = RAND_HASH(*pk*[2*i*], *pk*[2*i* + 1], *SEED*, *ADRS*).

   2) If *len′* mod 2 is equal to 1:

d) Set *pk*[floor(*len′*/ 2)] = *pk*[*len′* - 1].

   1) Set *len′* = ceil(*len′*/ 2).

   2) Increment *ADRS.treeHeight* by 1.

e) Return *pk*[0].

5.3.2.3 treeHash

The treeHash algorithm is an auxiliary function that computes a specific node of the tree. It is used in the context of generating key pairs and computing authentication paths. It makes use of the conventional stack data structure which has a push (insert an item at the top of the stack) and a pop (remove an item from the top of the stack) operation working in a last-in-first-out fashion.

Algorithm: treeHash(*SK, s, t, ADRS*).

Input: XMSS private key *SK*, an unsigned integer s representing the start index, an unsigned integer t representing the target node height, and an address *ADRS* formatted according to Table 1 as a Hash tree Address.

Output: n-byte root node of a sub-tree.

Steps:

a) If (s mod (2^t) is not equal to 0) return -1.

b) for i from 0 to 2^t - 1:

   1) Set *SEED* = *SK.SEED*.

   2) Set *ADRS.type* = 0.

   3) Set *ADRS.OTSAddress* = (s + i).

   4) Set *pk* = WOTS+_generate_pubkey(WOTS+_generate_privkey(*SK.SK_S, SEED, s + i, ADRS.layerAddress, ADRS.treeAddress*), *SEED, ADRS*).

   5) Set *ADRS.type* = 1

   6) Set *ADRS.ltreeAddress* = (s + i).

   7) Set *node* = L_TREE(*pk, SEED, ADRS*).

   8) Set *ADRS.type* = 2.

   9) Set *ADRS.padding* = 0.

   10) Set *ADRS.treeHeight* = 0.

   11) Set *ADRS.treeIndex* = (i + s).

   12) While (*Stack* is not empty and *Top* node on *Stack* has same height t' as *node*):

      i) Set *ADRS.treeIndex* = ((*ADRS.treeIndex* - 1) / 2).

      ii) Set *node* = RAND_HASH(*Stack*.pop(), *node*, *SEED*, *ADRS*).

      iii) Set *ADRS.treeHeight* = (*ADRS.treeHeight* + 1).

   13) *Stack*.push(*node*).

c) Return *Stack*.pop().

The description above assumes that the pseudo-random key generation method from 5.2.5.2.2 is used. Alternatively, if the method from 5.2.5.2.3 is used, then step b)4) would not reconstruct the private key, and instead it would use the one stored in *sk* (i.e. the WOTS+ private key).

5.3.3 XMSS Key Generation

5.3.3.1 General

This key generates an XMSS key pair for a given layer and tree. The XMSS private and public key generation methods described in 5.3.3.2 follow the definition provided in Section 7.2.1 of NIST SP 800-208.[12] The XMSS private and public keys consist of the following components:

NOTE *SEED* and *root* are public and stored in private and public keys.

— Private key:

  — If the pseudo-random method described in 5.2.5.2.2 is used, a seed *SK_S* (n bytes),

— If the random method described in 5.2.5.2.3 is used, 2^h WOTS+ private keys (2^h * *len* * *n* bytes),

— *SK_PRF*, a key for the PRF (*n* bytes),

— *idx*, index of the next WOTS+ private key that has not been used yet (8 bytes),

— *root*, the root of the tree (*n* bytes),

— *SEED*, the public seed (*n* bytes);

— Public key:

— *type*, the type code (4 bytes),

— *root*, the root of the tree (*n* bytes),

— *SEED*, the public seed (*n* bytes).

The XMSS_keygen algorithm also helps with the computation of the authentication path for the first signature. The *type* code, which uniquely identifies the algorithm and parameter configuration, which is returned as part of the public key, is assumed to be known by the implementation (as all other cipher parameters).

The XMSS key generation is given in two flavours: one that shall be used if XMSS is used isolated, and another one that shall be used if integrated into XMSS-MT key generation procedure. It is possible to implement both flavours as a single algorithm but this would lead to if-then-else branches and optional parameters. For the sake of clarity, they are presented separately.

Both methods assume that the pseudo-random WOTS+ private key generation method described in 5.2.5.2.2 is used, which derives all WOTS+ private key elements from *SK_S*. If the random method described in 5.2.5.2.3 is used, instead then the XMSS private key shall not have a *SK_S* element, and the XMSS private key shall store all ($2^{h/d}$ in total) WOTS+ private keys instead.

5.3.3.2 XMSS Key Generation (For XMSS-only)

Algorithm: XMSS_keygen()

Output: private key *SK*, public key *PK*.

Steps:

a) Let *SK.SK_S* be an *n*-byte string from the output generation function of an RBG.

b) Let *SK.SEED* be an *n*-byte string from the output generation function of an RBG.

c) Set *ADRS* = toByte(0, 32).

d) Let *SK_PRF* be an *n*-byte string from the output generation function of an RBG.

e) Set *root* = treeHash(*SK*, 0, *h, ADRS*).

f) Set *SK.idx = 0*.

g) Set *SK.root* = *root*.

h) Set *SK* = *idx* || *SK_S* || *SK_PRF* || *root* || *SK.SEED*.

i) Set *PK* = *type* || *root* || *SK.SEED*.

j) Return (*SK* || *PK*).

5.3.3.3 XMSS Key Generation (For XMSS-MT integration)

Algorithm: XMSS_keygen(*SK_MTS_S*, *L*, *T*, *ADRS*).

Input: an n-byte string from the output generation function of an RBG called SK_MTS_S, the layer L (in the range [0, …, d-1], where d is the number of layers for XMSS-MT), the tree index T, an n-byte ADRS.

Output: private key SK, public key PK.

Steps:

a) Let $SK.SEED$ be an n-byte string from the output generation function of an RBG.

b) Set $ADRS.layerAddress = L$.

c) Set $ADRS.treeAddress = T$.

d) Set SK.SK_S = $\mathrm{PRF}_{\mathrm{Keygen\_MT}}(SK\_MTS\_S, SEED \mid\mid ADRS)$.

e) Let SK_PRF be an n-byte string from the output generation function of an RBG.

f) Set $root$ = treeHash(SK, 0, h/d, $ADRS$).

g) Set $SK.idx = T * 2^{(h\ /\ d)}$.

h) Set $SK.root = root$.

i) Set $SK = idx \mid\mid SK\_S \mid\mid SK\_PRF \mid\mid root \mid\mid SK.SEED$.

j) Set $PK = type \mid\mid root \mid\mid SK.SEED$.

k) Return ($SK \mid\mid PK$).

5.3.4 XMSS Signing

Given an XMSS private key, a message and an authentication path, this algorithm generates an XMSS signature and updates the XMSS private key. This algorithm assumes that the authentication path has already been pre-computed using one of the authentication path computation algorithms. A simple authentication path computation algorithm is given in 5.3.5. It should be noted that the XMSS signing algorithm modifies the XMSS private key given the index update process.

The XMSS signature consists of the following components.

— idx_sig, the index of the used WOTS+ key pair (8 bytes),

— r, a byte string used for randomized message hashing (n bytes),

— sig_ots, a WOTS+ signature ($len \cdot n$ bytes),

— $AUTH$, the authentication path for the leaf ($h \cdot n$ bytes).

Algorithm: XMSS_sign(SK, $AUTH$, m).

Input: Private key SK, authentication path $AUTH$, message m.

Output: The updated private key SK and the signature Sig.

Steps:

a) Set $idx\_sig = SK.idx$.

b) Set $SK.idx = SK.idx + 1$.

c) Set $ADRS$ = toByte(0, 32).

d) Set r = PRF($SK.PRF$, toByte(idx_sig, 32)).

e) Set M' = H_msg($r \mid\mid SK.root \mid\mid$ toByte(idx_sig, n), m).

f) Set *ADRS.type* = 0.

g) Set *ADRS.OTSAddress* = *idx_sig*.

h) Set *sig_ots* = WOTS+_sign(WOTS+_generate_privkey(*SK.SK_S*, *idx_sig*, *ADRS.layerAddress*, *ADRS.treeAddress*), *M'*, *SK.SEED*, *ADRS*).

i) Set *Sig* = *idx_sig* || *r* || *sig_ots* || *AUTH*.

j) Return (*SK*, *Sig*).

In step b), *SK* shall be updated. No signature shall be returned if the update of *SK* fails.

5.3.5 XMSS Authentication Path Computation

This algorithm computes the authentication path given the XMSS private key *SK*, the WOTS+ key pair of index *i*, and *ADRS*. It is important to note that this algorithm is simple but not efficient. A more efficient variant called the BDS algorithm can be found in Reference [11]. Any authentication path algorithm, when correctly implemented, shall return the same authentication path for a given XMSS private key, WOTS+ key pair index and *ADRS* buffer.

Algorithm: buildAuth(*SK*, *i*, *ADRS*).

Input: the XMSS private key *SK*, the WOTS+ key pair index *i*, and *ADRS* which is 32-byte value formatted according to Table 1.

Output: *AUTH*, the authentication path for key pair index *i*.

Steps:

a) For *j* from 0 to *h*/*d* - 1:

   1) Set *k* = floor(*i* / (2^j)) XOR 1.

   2) Set *AUTH*[*j*] = treeHash(*SK*, *k* * 2^j, *j*, *ADRS*).

b) Return *AUTH*.

5.3.6 XMSS Verification

5.3.6.1 General

The XMSS verification algorithm is divided into two procedures: the first computes a root node from the XMSS signature and the second one checks if the produced root node matches the XMSS public key root node. If they match, the signature is accepted as authentic, and rejected otherwise.

5.3.6.2 Compute Root Node from Signature

This algorithm computes a root node from a signature, a message, *PK*, and an *n*-byte *ADRS*.

Algorithm: XMSS_rootFromSig(*sig*, *M'*, *PK*, *ADRS*).

Input: Tree signature *sig*, *n*-byte *ADRS* message *M'*, public key *PK*.

Output: The *root* node computed from the signature.

Steps:

a) Set *sig_ots* = *sig.sig_ots*.

b) Set *idx_sig* = *sig.idx_sig*.

c) Set *AUTH* = *sig.auth*.

d) Set *SEED* = *PK.SEED*.

e) Set *ADRS.Type* = 0.

f) Set *ADRS.OTSAddress* = *idx_sig*.

g) Set *pk_ots* = WOTS+_pk_from_sig(*sig_ots, M', SEED, ADRS*).

h) Set *ADRS.Type* = 1.

i) Set *ADRS.ltreeAddress* = *idx_sig*.

j) Set *node*[0] = L_TREE(*pk_ots, SEED, ADRS*).

k) Set *ADRS.Type* = 2.

l) Set *ADRS.Padding* = 0.

m) Set *ADRS.TreeIndex* = *idx_sig*.

n) For *i* from 0 to *h*/*d* - 1:

   1) Set *ADRS.TreeHeight* = *i*.

   2) If floor(*idx_sig* / 2^i) mod 2 is equal to 0:

      i) Set *ADRS.Treeindex* = *ADRS.TreeIndex* / 2.

      ii) Set *node*[1] = RAND_HASH(*node*[0], *AUTH*[*i*], *SEED, ADRS*).

   3) If floor(*idx_sig* / 2^i) mod 2 is not equal to 0:

      i) Set *ADRS.Treeindex* = (*ADRS.TreeIndex* - 1) / 2.

      ii) Set *node*[1] = RAND_HASH(*AUTH*[*i*], *node*[0], *SEED*, *ADRS*).

   4) Set *node*[0] = *node*[1].

o) Return *node*[0].

5.3.6.3 XMSS Verify

This algorithm verifies an XMSS signature using the XMSS public key and a message.

Algorithm: XMSS_verify(*Sig, PK*, *m*).

Input: Signature *Sig*, public key *PK*, message *m*.

Output: *VALID* or *INVALID*.

Steps:

a) Set *ADRS* = toByte(0, 32).

b) Set *M'* = H_msg(*r* || *PK.root* || toByte(*idx_sig*, *n*), *m*);

c) Set *node* = XMSS_rootFromSig(*Sig, M', PK*, *ADRS*).

d) If *node* is equal to *PK.Root*, then return *VALID*.

e) If *node* is not equal to *PK.Root*, then return *INVALID*.

5.4 XMSS-MT Algorithms

5.4.1 General

The XMSS Multi-Tree (XMSS-MT) scheme is a variant of the XMSS scheme and allows for a very large number of signatures (e.g. 2^{60}). To achieve this, the Merkle tree used in XMSS-MT is composed of many sub-trees. Each sub-tree has a height h / d, and there are d layers of sub-trees. The parameter d is important for XMSS-MT, while it can be assumed as d = 1 for XMSS.

The type code, which uniquely identifies the algorithm and parameter configuration (returned as part of the public key), is assumed to be known by the implementation. This is also the case for all other cipher parameters).

5.4.2 XMSS-MT key Generation

This algorithm generates an XMSS-MT private and public key.

Algorithm: XMSS_MT_keygen().

Input: No input.

Output: An XMSS-MT key pair (*SK_MT, PK_MT*).

Steps:

a) Set *SK_MT.idx_MT* = 0.

b) Set *SK_MT.idx* = 0.

c) Let *SK_MT.SK_S* be an *n*-byte string from the output generation function of an RBG.

d) Let *SK_MT.SK_PRF* be an *n*-byte string from the output generation function of an RBG.

e) Let *SK_MT.SEED* be an *n*-byte string from the output generation function of an RBG.

f) Set *ADRS* = toByte(0, 32)

g) For *layer* from 0 to *d* - 1:

   1) Set *ADRS.layerAddress* = *layer*.

   2) For *tree* from 0 to (1 << ((*d* - 1 - *layer*) * (*h* / *d*))) - 1:

      i) Set *ADRS.treeAddress* = *tree*.

      ii) Set (*XMSS_SK*, *XMSS_PK*) = XMSS_keygen(*SK_MT.SK_S*, *tree*, *layer*, *ADRS*)

      iii) setXMSS_SK(*SK_MT, XMSS_SK.SK_S, tree, layer*).

h) Set *XMSS_SK* = getXMSS_SK(*SK_MT*, 0, *d*-1).

i) Set *XMSS_SK.SEED* = *SK_MT.SEED*.

j) Set *root* = treeHash(*XMSS_SK*, 0, *h* / *d, ADRS*).

k) Set *SK_MT.root* = *root*.

l) Set *PK_MT* = *type* || *root* || *SEED*.

m) Return (*SK_MT* || *PK_MT*).

setXMSS_SK(*SK_MT, SK_MT_S, tree, layer*) denotes the procedure that sets the private key for the specific XMSS subtree. getXMSS_SK(*SK_MT, tree, layer*) denotes the procedure that retrieves the specific subkey.

5.4.3 XMSS-MT signing

5.4.3.1 XMSS-MT treeSig

The treeSig algorithm generates a WOTS+ signature for a message with a given authentication path.

Algorithm: treeSig(*M′*, *SK*, *idx_sig*, *ADRS*)

Input: *n*-byte message *M′*, XMSS private key *SK*, signature index *idx_sig*, *ADRS*

Output: WOTS+ signature *sig_ots* and authentication path *auth*

Steps:

a) Set *auth* = buildAuth(*SK, idx_sig, ADRS*).

b) Set *ADRS.type* = 0.

c) Set *ADRS.OTSAddress* = *idx_sig*.

d) *sig_ots* = WOTS+_sign(WOTS+_generate_privkey*(SK.SK_S, idx_sig, ADRS.layerAddress, ADRS.treeAddress), M′, SK.SEED, ADRS*).

e) Return *sig_ots* || *auth*.

5.4.3.2 XMSS-MT Signing

This algorithm generates an XMSS-MT signature and updates the XMSS-MT private key. This algorithm requires an authentication path computation algorithm. A simple authentication path computation algorithm is provided in 5.3.5. It should be noted that the XMSS-MT signing algorithm modifies the private key given the index update process. The XMSS-MT signing algorithm uses the treeSig algorithm as an intermediate step.

Algorithm: XMSS_MT_sign(*SK_MT, m*).

Input: Private key *SK_MT*, message *m*.

Output: Updated private key *SK_MT* and Signature *Sig_MT*.

Steps:

a) Set *ADRS* = toByte(0, 32).

b) Set *SEED* = *SK_MT.SEED*.

c) Set *SK_PRF* = *SK_MT.SK_PRF*.

d) Set *idx_sig* = *SK_MT.idx*.

e) Set *SK_MT.idx* = *idx_sig* + 1.

f) Set *r* = PRF(*SK_PRF*, toByte(*idx_sig*, 32)).

g) Set *M′* = H_msg(*r* || *SK_MT.root* || toByte(*idx_sig*, *n*), *m*).

h) Set *Sig_MT* = *idx_sig*.

i) Let *idx_tree* be the (*h* - (*h* / *d*)) most significant bits of *idx_sig*.

j) Let *idx_leaf* be the (*h* / *d*) least significant bits of *idx_sig*.

k) Set *SK* = *idx_leaf* || getXMSS_SK(*SK_MT*, *idx_tree*, 0) || *SK_PRF* || toByte(0, *n*) || *SEED*.

l) Set *ADRS.layerAddress* = 0.

m) Set *ADRS.treeAddress* = *idx_tree*.

n) Set *Sig_tmp* = treeSig(*M′*, *SK*, *idx_leaf*, *ADRS*).

o) Set *Sig_MT* = *Sig_MT* || *r* || *Sig_tmp*.

p) For *j* from 1 to *d* - 1:

   1) Set *root* = treeHash(*SK*, *0*, *h* / *d*, *ADRS*).

   2) Let *idx_leaf* be the (*h* / *d*) least significant bits of *idx_tree*.

   3) Let *idx_tree* be the (*h* - *j* * (*h* / *d*)) most significant bits of *idx_tree*.

   4) Set *XMSS_SK* = getXMSS_SK(*SK_MT*, *idx_tree*, *j*).

   5) Set *SK* = *idx_leaf* || *XMSS_SK* || *XMSS_SK.SK_PRF* || toByte(0, *n*) || *SEED*.

   6) Set *ADRS.layerAddress* = *j*.

   7) Set *ADRS.treeAddress* = *idx_tree*.

   8) Set *Sig_tmp* to be treeSig(*root*, *SK*, *idx_leaf*, *ADRS*).

   9) Set *Sig_MT* to be *Sig_MT* || *Sig_tmp*.

q) Return *SK_MT* || *Sig_MT*.

It should be noted that signatures produced in step p) can be cached and only computed once.

5.4.4 XMSS-MT Verification

This algorithm verifies if an XMSS-MT signature is authentic.

Algorithm: XMSS_MT_verify(*Sig_MT*, *PK_MT*, *m*).

Input: Signature *Sig_MT*, public key *PK_MT*, message *m*.

Output: *VALID* or *INVALID*.

Steps:

a) Set *idx_sig* = *Sig_MT.idx*.

b) Set *SEED* = *PK_MT.SEED*.

c) Set *ADRS* = toByte(0, 32).

d) Set *M′* = H_msg(*Sig_MT.R* || *PK_MT.Root* || (toByte(*idx_sig*, *n*)), *m*).

e) Let *idx_leaf* be the (*h* / *d*) least significant bits of *idx_sig*.

f) Let *idx_tree* be the (*h* - *h* / *d*) most significant bits of *idx_sig*.

g) Set *Sig′* = getXMSSSignature(*Sig_MT*, 0).

h) Set *ADRS.layerAddress* = 0.

i) Set *ADRS.treeAddress* = *idx_tree*.

j) Set *Sig′.idx* = *idx_leaf*.

k) Set *node* = XMSS_rootFromSig(*Sig′*, *M′*, *PK_MT*, *ADRS*).

l) For *j* from 1 to *d* - 1:

   1) Set *idx_leaf* to be the (*h* / *d*) least significant bits of *idx_tree*.

2) Set *idx_tree to* be the (h - $j*h/d$) most significant bits of *idx_tree*.

3) Set *Sig'* = getXMSSSignature(*Sig_MT, j*).

4) Set *ADRS.layerAddress* = *j*.

5) Set *ADRS.treeAddress* = *idx_tree.*

6) Set *Sig'.idx* = *idx_leaf.*

7) Set *node* = XMSS_rootFromSig(*Sig', node, PK_MT, ADRS*).

m) If node is equal to *PK_MT.root*, then return *VALID*.

n) If node is not equal to *PK_MT.root*, then return *INVALID*.

getXMSSSignature(*Sig_MT, i*) denotes the function that returns the *i*-th XMSS signature from an XMSS-MT signature.

5.5 Suggested parameters

This document provides sets of parameters based on SHA2-256, SHA2-256/192 (SHA2-256 truncated to the most significant 24 bytes), or SHAKE256 with an output size of n bytes. The output digest length is represented as a parameter n which is given in bytes, i.e. $n = 24$ or $n = 32$.

Besides having different choices for the underlying hash function and the output length, they also differ in terms of h (height of the tree), which can be: 10, 16 or 20, for XMSS, and 20, 40, 60 for XMSS-MT, respectively, and d (the number of multi-trees in XMSS-MT). All XMSS and XMSS-MT parameters use a single value for the Winternitz parameter, $w = 16$, to be consistent with other XMSS published standards.

The parameters for XMSS and XMSS-MT, including their type code and corresponding private key, public key and signature sizes, are listed in Table 2 and Table 3, respectively.

Table 2 — XMSS sizes (in bytes)

Type Code for SHA2-256	Type Code for SHAKE256	n	h	w	Private key size ($4n + 8$)	Public key size ($2n + 4$)	Signature size ($8+n+ (len+h) * n$)
0x00000001	0x00000010	32	10	16	**136**	**68**	**2 504**
0x00000002	0x00000011	32	16	16	**136**	**68**	**2 696**
0x00000003	0x00000012	32	20	16	**136**	**68**	**2 824**
0x0000000D	0x00000013	24	10	16	**104**	**52**	**1 496**
0x0000000E	0x00000014	24	16	16	**104**	**52**	**1 640**
0x0000000F	0x00000015	24	20	16	**104**	**52**	**1 736**

Table 3 — XMSS-MT sizes (in bytes)

Type Code for SHA2-256	Type Code for SHAKE256	n	h	d	w	Private key size ceil(h/8) + $3n + 4$	Public key size $2n + 4$	Signature size ceil(h / 8) + n + ($h + d * len$) * n
0x00000001	0x00000029	32	20	2	16	**103**	**68**	**4 963**
0x00000002	0x0000002A	32	20	4	16	**103**	**68**	**9 251**
0x00000003	0x0000002B	32	40	2	16	**105**	**68**	**5 605**
0x00000004	0x0000002C	32	40	4	16	**105**	**68**	**9 893**
0x00000005	0x0000002D	32	40	8	16	**105**	**68**	**18 469**
0x00000006	0x0000002E	32	60	3	16	**109**	**68**	**8 392**
0x00000007	0x0000002F	32	60	6	16	**109**	**68**	**14 824**
0x00000008	0x00000030	32	60	12	16	**109**	**68**	**27 688**
0x00000021	0x00000031	24	20	2	16	**79**	**52**	**2 955**

Table 3 *(continued)*

Type Code for SHA2-256	Type Code for SHAKE256	n	h	d	w	Private key size ceil(h/8) + 3n + 4	Public key size 2n + 4	Signature size ceil(h / 8) + n + (h + d * len) * n
0x00000022	0x00000032	24	20	4	16	**79**	**52**	**5 403**
0x00000023	0x00000033	24	40	2	16	**81**	**52**	**3 437**
0x00000024	0x00000034	24	40	4	16	**81**	**52**	**5 885**
0x00000025	0x00000035	24	40	8	16	**81**	**52**	**10 781**
0x00000026	0x00000036	24	60	3	16	**85**	**52**	**5 145**
0x00000027	0x00000037	24	60	6	16	**85**	**52**	**8 817**
0x00000028	0x00000038	24	60	12	16	**85**	**52**	**16 161**

6 LMS and HSS schemes

6.1 Byte ordering convention

The functions u8str(x), u16str(x), and u32str(x) convert an unsigned integer to a sequence of bytes of length 1, 2, and 4, respectively. The reverse operations are defined with the respective functions strtou8(x), strtou16(x), and strtou32(x).

6.2 Converting to base 2^W

Algorithm: coef(x, i, W)

Input: A byte string x, index i and base W in {1, 2, 4, 8}.

Output: The ith W-bit value of x, when interpreting x as a sequence of W-bit values.

Steps:

a) Set $tmp = x[\text{floor}(i * W / 8)]$. Here, $x[j]$ is the j-th byte from x.

b) Set $tmp = tmp >> (8 - (W * (i \bmod (8 / W)) + W))$.

c) Return $(2^W - 1)$ & tmp.

6.3 Checksum Calculation

Algorithm: checksum(x, W)

Input: A byte string x of length l and base 2^W.

Output: The checksum for x as a 16-bit unsigned integer.

Steps:

a) Set $sum = 0$

b) Set $len_1 = \text{ceil}(8*l/W)$

c) Set $len_2 = \text{ceil}((\text{floor}(\text{lb}((2^W - 1) * len_1)) + 1) / W)$

d) Set $ls = 16 - (len_2 * W)$

e) For i from 0 to $(l * 8 / W) - 1$, set $sum = sum + (2^W - 1) - \text{coef}(x, i, W)$.

f) Return $(sum << ls)$.

6.4 Type code

A type code is an unsigned integer that is associated with a particular format of LM-OTS, LMS, and HSS. All signatures and public keys use a 4-byte typecode which specifies the precise details of the format. The typecode specifies the values for n and W, and the hash function to use for H.

6.5 LM-OTS

6.5.1 General

6.5 defines the LM-OTS signature scheme. LM-OTS is a one-time signature scheme used as a building block for both LMS and HSS. LM-OTS shall not be used outside the context of LMS or HSS. LM-OTS has the following parameters:

— n: Output size of the hash function in bytes. This shall be 24 or 32.

— W: This shall be 1, 2, 4 or 8. It should be noted that in LMS/HSS, this parameter is the log of the same parameter used in XMSS.

— H: The cryptographic hash function used in LM-OTS. This shall be SHA2-256 (Dedicated Hash-Function 4 defined in ISO/IEC 10118-3), SHA256/192 (SHA2-256 truncated to the most significant 24 bytes) or SHAKE256 with an output size of n (see ISO/IEC 10118-3:2018, C.2).

The parameter n determines the security of LM-OTS. The parameter W determines the length of the hash chains in the scheme, which influences the signature size and computation time.

The LM-OTS key generation, signing and verification further take the following parameters into account:

— I: This is a 16-byte identifier which indicates which Merkle tree this LM-OTS is used with.

— q: This is a 32-bit integer that indicates the leaf of the Merkle tree where the LM-OTS public key is used.

— len: The number of n-byte string elements that make up an LM-OTS signature.

The value len is fully determined by n and W, and computed as follows:

— len_1 = ceil(8*n/W).

— len_2 = ceil((floor(lb((2^W - 1) * len_1)) + 1) / W).

— $len = len_1 + len_2$.

A unique parameter set name is associated with each parameter set. For example, LMOTS_SHA256_N32_W4 refers to LM-OTS using SHA2-256 with n = 32 and W = 4.

6.5.2 Key generation

6.5.2.1 Private Key

Algorithm: LMOTS_generate_privkey(I, q, $type$).

Input: An identifier I, the leaf index q, the 4-byte typecode $type$.

Output: LM-OTS private key.

Steps:

a) Generate len n-byte strings x[0], ..., x[len-1] from the output generation function of an RBG.

b) Output private key k = u32str($type$) || I || u32str(q) || x[0] || x[1] || ... || x[len-1].

An LM-OTS private key shall be used to sign at most one message.

6.5.2.2 Private Key (from seed)

Alternatively, a private key can be generated from an *n*-byte seed *SK.S*.

Algorithm: LMOTS_generate_privkey_from_seed(*I, q, SK.S, type*).

Input: An identifier *I*, the leaf index *q*, the *n*-byte seed *SK.S*, the 4-byte typecode *type*.

Output: LM-OTS private key.

Steps:

a) for *i* from 0 to *len* - 1:

   1) $x[i]$ = H(*I* || u32str(*q*) || u16str(*i*) || u8str(`0xff`) || *SK.S*).

b) Return private key *k* = u32str(*type*) || *I* || u32str(*q*) || $x[0]$ || $x[1]$ || ... || $x[len\text{-}1]$.

An LM-OTS private key shall be used to sign at most one message.

6.5.2.3 Public Key

The LM-OTS public keys are generated from the private key in the following way:

Algorithm: LMOTS_generate_pubkey(*k*).

Input: LM-OTS private key *k*.

Output: LM-OTS public key.

Steps:

a) Extract the values *I*, *q*, *type* and *x*[] from *k*.

b) For *i* from 0 to *len* - 1:

   1) Set *tmp* = $x[i]$.

   2) For *j* from 0 to 2^W - 2, set *tmp* = H(*I* || u32str(*q*) || u16str(*i*) || u8str(*j*) || *tmp*).

   3) Set $y[i]$ = *tmp*.

c) Set *K* = H(*I* || u32str(*q*) || u16str(`0x8080`) || $y[0]$ || ... || $y[len\text{-}1]$).

d) Return u32str(*type*) || *I* || u32str(*q*) || *K*.

6.5.3 Signing

Algorithm: LMOTS_sign(*m, k*).

Input: Message *m* to be signed, LM-OTS private key *k*.

Output: Signature *s* for message *m*.

Steps:

a) Extract the values *I*, *q, x* and *type* from *k*.

b) Let *C* be an *n*-byte string from the output generation function of an RBG.

c) Set *M'* = H(*I* || u32str(*q*) || u16str(`0x8181`) || *C* || *m*).

d) For *i* from 0 to *len* - 1:

   1) Set *a* = coef(*M'* || checksum(*M'*,*W*).

2) Set $tmp = x[i]$.

3) For j from 0 to a - 1, set tmp = H(I || u32str(q) || u16str(i) || u8str(j) || tmp).

4) Set $y[i] = tmp$.

e) Return u32str($type$) || C || y[0] || ... || y[len-1].

6.5.4 Verification

Verifies if a signature is authentic given a signature, a public key and a message. The LM-OTS verification process is divided into two parts. The first is to recompute the LM-OTS public key from the LM-OTS signature. The second is to check if the recomputed LM-OTS public key matches the original one.

When used in the context of LMS or HSS signature verification, only the first part of the LM-OTS verification process is performed, i.e. the LM-OTS public key recovery step. It is never compared with any other LM-OTS public key. This verification is implicitly done at the LMS or HSS signature verification level.

Algorithm: LMOTS_pubkey_from_sig(m, s, pub_key).

Input: Message m, signature s, the LMS public key pub_key.

Output: The recomputed LM-OTS public key Kc.

Steps:

a) Parse s, to obtain C and y[0], ..., y[len - 1]. Parse pub_key to obtain I and q.

b) Set M' = H(I || u32str(q) || u16str(`0x8181`) || C || m).

c) For i from 0 to len - 1:

1) Set a = coef(M' || checksum(M',W).

2) Set $tmp = y[i]$.

3) For j from a to 2^W - 2, set tmp = H(I || u32str(q) || u16str(i) || u8str(j) || tmp).

4) Set $z[i] = tmp$.

d) Set Kc = H(I || u32str(q) || u16str(`0x8080`) || z[0] || z[1] || ... || z[len-1]).

e) Return Kc.

6.5.5 Suggested Parameters

The suggested parameters for LM-OTS can be found in Table 4.

Table 4 — LM-OTS parameter sets

Type Code for SHA2	Type Code for SHAKE	n	W
0x00000001	0x00000009	32	1
0x00000002	0x0000000A	32	2
0x00000003	0x0000000B	32	4
0x00000004	0x0000000C	32	8
0x00000005	0x0000000D	24	1
0x00000006	0x0000000E	24	2
0x00000007	0x0000000F	24	4
0x00000008	0x00000010	24	8

6.6 LMS

6.6.1 General

The LMS scheme can sign a fixed number of messages. It uses the LM-OTS signature scheme defined in 6.5 and builds a full binary Hash tree where the leaves are the public keys of LM-OTS key pairs. Each node of this tree is associated with a node number. The root is defined to be the node with number 1. For a node with the number N, the left child is defined as the node with number $2*N$ and the right child as the node with number $2*N + 1$.

LMS uses the following parameters:

— h: The height of the tree. This shall be in {5, 10, 15, 20, 25}.

— m: The number of bytes associated with each node. This shall be in {24, 32}. This shall not be confused with the message m.

— H: The cryptographic hash function used in LM-OTS and to generate the Merkle tree nodes in LMS. This shall be SHA2-256, SHA2-256/192 (SHA2-256 truncated to the most significant 24 bytes) or SHAKE256 with an output size of m.

— OTS: An LM-OTS signature scheme with the parameter defined in 6.5. The hash function used in this parameter set shall be the same as in LMS.

An LMS tree will therefore have 2^h leaves, which allows it to sign up to 2^h messages. A parameter set is defined with a unique name LMS_SHA256_M32_H10, which corresponds to an LMS tree using SHA2-256 with $m = 32$ and $h = 10$.

The OIDs for these algorithms shall be in accordance with Annex A. Test vectors can be found in Annex C.

6.6.2 Key generation

The LMS private and public keys are composed of:

Private key:

— q, the leaf index (8 bytes).

— *type*, the type code (4 bytes) .

— *otstype*, the type code for the OTS (4 bytes).

— I, the identifier (16 bytes).

— If the method described in 6.5.2.2 is used, a seed SK_S (n bytes)

— If the method described in 6.5.2.1 is used, 2^h LM-OTS private keys ($2^h * len * n$ bytes).

An LMS private key consists of 2^h LM-OTS private keys (or, alternatively, a seed SK_S which is used to derive these LM-OTS private keys); an index q, which indicates which private LM-OTS key should be used next; and the identifier I. The value q shall be initialized to 0. Each LM-OTS key shall be generated with a different value q. LM-OTS are indexed sequentially from 0 to 2^h - 1. The LM-OTS key pairs used here are referred to as ($LM\_OTS\_PUB\_KEY[i]$, $LM\_OTS\_PRIVATE\_KEY[i]$), for i in 0,...,2^h - 1. Set I to be 16 bytes output from an RBG.

Public key:

— *type*, the type code (4 bytes) .

— *otstype*, the type code for the OTS (4 bytes).

— I, the identifier (16 bytes).

— $T[1]$, the root element of the tree (m bytes).

An LMS public key is defined as the root of the binary Hash tree. In order to compute the root, the following process shall be used. The string for the N-th node is denoted here as $T[N]$, and the nodes are indexed from 1 to $2^{(h+1)} - 1$:

a) If a node is a leaf node (this means $N >= 2^h$), then the value corresponding to the node is:

H(I || u32str(N) || u16str(`0x8282`) || get K component from within concatenated string $LM\_OTS\_PUB\_KEY[N - 2^h]$)

b) Otherwise the node is computed as:

H(I || u32str(N) || u16str(`0x8383`) || $T[2*N]$ || $T[2*N + 1]$)

c) Output the LMS public key as:

u32str(*type*) || u32str(*otstype*) || I || $T[1]$

6.6.3 Signing

An LMS signature consists of three parts:

— The number q corresponding to the leaf index used in this signature. This is a 4-byte value.

— An LM-OTS signature and the typecode indicating which LMS algorithm is used.

— An array of h m-byte values, which are the authentication path.

The authentication path is the minimal number of nodes required, such that someone verifying the signature can recompute the root of the binary Hash tree from the authentication path and the LM-OTS signature.

Algorithm: LMS_sign(m, k).

Input: Message m to be signed, LMS private key k.

Output: Signature s for message m.

Steps:

a) Let *q_sig* be the current index q.

b) Update index q to the next unused index.

c) Compute an LM-OTS signature of the message m, using the LM-OTS key pair corresponding to the index *q_sig* from the LMS private key k.

d) Compute the authentication path.

e) Set s = u32str(*q_sig*) || *lmots_signature* || u32str(*type*) || *path*[0] || *path*[1] || *path*[2] || ... || *path*[h-1]

Any implementation of LMS_sign shall ensure that the index q is updated before a signature is released.

6.6.4 Verification

Algorithm: LMS_verify(*m, public_key, s*).

Input: Signature s, message m to be verified, LMS public key *public_key*.

Output: *VALID* if the signature is correct, otherwise *INVALID*.

Steps:

a) Let *lms_type* be the type in *public_key*.

b) Let *sig_type* be the type in s.

c) If *lms_type* does not equal *sig_type*, return *INVALID*.

d) Extract the LM-OTS signature s_{ots} using the index q from s, and compute the value Kc using LMOTS_pubkey_from_sig(m, s_{ots}, $public\_key$).

e) Compute the root node candidate Tc for the LMS tree in the following way:

1) Set $node\_num = 2^h + q$.

2) Set tmp = H(I || u32str($node\_num$) || u16str(`0x8282`) || Kc).

3) Set $i = 0$.

4) while $node\_num > 1$ do:

i) $parent\_node\_num$ = floor($node\_num$ / 2)

ii) If $node\_num$ is odd, set

tmp = H(I || u32str($parent\_node\_num$) || u16str(`0x8383`) || $path[i]$ || tmp).

iii) If $node\_num$ is even, set

tmp = H(I || u32str($parent\_node\_num$) || u16str(`0x8383`) || tmp || $path[i]$).

iv) Set $node\_num = parent\_node\_num$.

v) Set $i = i + 1$.

5) Set $Tc = tmp$.

f) If Tc is equal to the value $T[1]$ in the public key, then the output is *VALID*. Otherwise, the signature shall be rejected and the output is *INVALID*.

6.6.5 Suggested Parameters

The suggested parameters for LMS can be found in Table 5.

Table 5 — LMS sizes (in bytes)

Type Code for SHA2	Type Code for SHAKE	n	h
`0x00000005`	`0x0000000F`	32	5
`0x00000006`	`0x00000010`	32	10
`0x00000007`	`0x00000011`	32	15
`0x00000008`	`0x00000012`	32	20
`0x00000009`	`0x00000013`	32	25
`0x0000000A`	`0x00000014`	24	5
`0x0000000B`	`0x00000015`	24	10
`0x0000000C`	`0x00000016`	24	15
`0x0000000D`	`0x00000017`	24	20
`0x0000000E`	`0x00000018`	24	25

6.7 HSS

6.7.1 General

Using a very large number of nodes in a tree is costly, owing to the effort for generating a key pair scales with the size of the tree. If a large number of signatures are required to be supported, the hierarchical signature scheme (HSS) can be used.

HSS uses the following parameters:

— *L*: The number of layers of LMS trees used. *L* shall be between 1 and 8, inclusive.

All layers shall use the same hash function. For each layer, the same LMS and LM-OTS parameter set shall be used. Different layers may use a different set of LMS and LM-OTS parameters.

The OIDs for these algorithms shall be in accordance with Annex A. Test vectors can be found in Annex C.

6.7.2 Key generation

The HSS private and public keys are composed of:

Private key:

— *q*, the leaf index (8 bytes).

— u32str(*L*), the number of layers (4 bytes).

— *type*, the type code (4 bytes) .

— *otstype*, the type code for the OTS (4 bytes).

— *I*, the identifier (16 bytes).

— If the method described in 6.5.1.2 is used, a seed *SK_S* (*n* bytes)

— If the method described in 6.5.1.1 is used, 2^h LM-OTS private keys (2^h * *len* * *n* bytes).

Public key:

— u32str(*L*), the number of layers (4 bytes).

— *type*, the type code (4 bytes) .

— *otstype*, the type code for the OTS (4 bytes).

— *I*, the identifier (16 bytes).

— *T*[1], the root element of the tree (*m* bytes).

If different LMS and LM-OTS parameters are used across the layers, then the private and public key shall contain the *type* and *otstype* for each individual layer.

An HSS key pair is generated in the following way:

a) Generate an LMS key pair. The corresponding private key is *prv*[0] and public key *pub*[0].

b) For each *i* from 1 to *L* - 1.

   1) Generate an LMS key pair. The corresponding private key is assigned to *prv*[*i*] and the public key to *pub*[i].

   2) Sign *pub*[*i*] using *prv*[*i* - 1] with LMS. This computes the signature *sig*[*i* - 1] = LMS_sign(*pub*[*i*], *prv*[*i* - 1]).

The HSS private key consists of the values *prv*[0], ..., *prv*[*L*-1], *pub*[0], ..., *pub*[*L*-1], and *sig*[0], ..., *sig*[*L*-2]. It is not necessary to keep secret the values *pub* and *sig*. The values *prv*[1],..., *prv*[*L*-1] and *pub*[1],..., *pub*[*L*-1] are updated during signature generation.

The public key of HSS is the root node of the LMS tree at layer 0 and the number of layers: u32str(*L*) || *type* || *otstype* || *I* || *pub*[0].

6.7.3 Signing

To sign a message *m* with HSS, the following steps are completed:

a) Check if the LMS key *prv*[*L*-1] can still sign messages:

1) If it can still sign messages, then sign *m* with the LMS key and set *sig*[*L*-1] to the value of the signature.

2) If the number of signatures for the LMS key *prv*[*L*-1] is exhausted, find the smallest value *d* such that *prv*[*d*], *prv*[*d*+1], ..., *prv*[*L*-1] are exhausted.

i) If *d* is equal to 0, then the HSS key pair is exhausted and no further signature shall be generated with this HSS key pair.

ii) If *d* is not equal to 0, then new LMS key pairs are generated for each layer *i* from *d* to *L*-1 as follows:

(A) Generate a new LMS key pair and assign it to *prv*[*i*] and *pub*[*i*].

(B) Sign the value *pub*[*i*] with *prv*[*i* - 1], and set *sig*[*i* - 1] to the resulting value.

iii) Sign *m* with *prv*[*L* - 1] and set *sig*[*L*-1] to the resulting value.

b) The HSS signature is then defined as:

$$\mathrm{u32str}(L - 1) \mid\mid sig[0] \mid\mid pub[1] \mid\mid \ldots \mid\mid pub[L\text{-}1] \mid\mid sig[L\text{-}1]$$

The algorithm above assumes that the LMS signing algorithm updates the state *prv*, thus preventing key reuse.

6.7.4 Verification

To verify a signature *S*, for a message *M*, with the public key *pubkey*, perform the following steps:

a) Set *Nspk* = first four bytes of *S*.

b) If *Nspk*+1 is not equal to the number of levels *L* in pub:

1) return *INVALID*.

c) For (*i* = 0; *i* < *Nspk*; *i* = *i* + 1):

1) Set *siglist*[*i*] = next LMS signature parsed from *S*.

2) Set *publist*[*i*] = next LMS public key parsed from *S*.

d) Set *siglist*[*Nspk*] = next LMS signature parsed from *S*.

e) Set *key* = *pub*.

f) For (*i* = 0; *i* < *Nspk*; *i* = *i* + 1)

1) Set *sig* = *siglist*[*i*].

2) Set *msg* = *publist*[*i*].

3) If (lms_verify(*msg*, *key*, *sig*) is not *VALID*):

i) return *INVALID*.

4) Set *key* = *msg*.

g) return lms_verify(*M*, *key*, *siglist*[*Nspk*]).

6.7.5 Suggested Parameters

Each LMS tree in an HSS instance shall use a parameter set from Table 5. Additionally, all LMS instances shall use the same hash function. A specific level in an HSS instance shall use the same LMS and LM-OTS parameter sets. Different LMS and LM-OTS may be used on different levels, as long as the same hash function is used.

7 State management

One of the main challenges for the deployment of stateful hash-based signatures schemes refers to the process of state management, i.e. the ability to ensure that a signature state is never re-used. This is necessary for security purposes. The stateful HBS schemes defined in this document are based on one-time signature (OTS) building blocks which lose their security guarantees if the same OTS private key is used more than once (see Reference [7] for a comprehensive assessment of the security impact of reusing the same OTS private key). The problem of state management is well-known in the literature (see Reference [8]), and therefore this clause presents requirements for practitioners to implement robust state management mechanisms.

a) The state used by the stateful HBS schemes defined in this document is a piece of information that shall be stored, maintained and otherwise updated for the whole lifespan of the private key. Therefore, the state shall be stored in a secure non-volatile memory region inside the signing module. A hardware security module (HSM) is recommended to be used for this purpose to prevent users' access to the state and make the process of updating the state transparent to the signers.

b) Once a signature request is received, the signer shall first update the state and only then start the signing procedure. If this process was done in reverse order, there is a risk that the signature is produced but the state remains in its previous value (for example, due to a fault in the equipment occurring right after the signature is produced). In other words, the ultimate goal is ensuring that these two processes are performed in an atomic fashion.

Other mechanisms to prevent state re-use may be used.

Annex A
(normative)

Object identifiers and ASN.1 module

This annex lists the object identifiers assigned to the digital signature mechanisms specified in this document, namely: XMSS, XMSS-MT, LMS, and HSS. It should be noted that the additional information required for interoperability such as the selected parameter set or the selected hash function are given in the *type* field, which is part of the public key and signature data structures as specified in 5.5, 6.5.5, and 6.6.5.

```
DigitalSignatureWithAppendixStatefulHash
  {iso(1) standard(0) digital-signature-with-appendix (14888) part4(4)
  asn1-module(1) stateful-hash-based-mechanisms(0) version1(1)}

DEFINITIONS EXPLICIT TAGS ::= BEGIN

id-isoiec14888-4 OBJECT IDENTIFIER ::=
  {iso(1) standard(0) digital-signature-with-appendix(14888) part4(4)}
id-14888-4-algorithms OBJECT IDENTIFIER ::= {id-isoiec14888-4 algorithm(0)}

id-dswa-sfh-xmss OBJECT IDENTIFIER ::= { id-14888-4-algorithms 1 }
id-dswa-sfh-xmssmt OBJECT IDENTIFIER ::= { id-14888-4-algorithms 2 }
id-dswa-sfh-lms OBJECT IDENTIFIER ::= { id-14888-4-algorithms 3 }
id-dswa-sfh-hss OBJECT IDENTIFIER ::= { id-14888-4-algorithms 4 }

alg-dswa-sfh-xmss ALGORITHM ::= {
  PARMS         Param-dswa-sfh-xmss
  IDENTIFIED BY { id-dswa-sfh-xmss } }

Param-dswa-sfh-xmss ::= OCTET STRING (SIZE (4)) -- xmss-id

alg-dswa-sfh-xmssmt ALGORITHM ::= {
  PARMS         Param-dswa-sfh-xmssmt
  IDENTIFIED BY { id-dswa-sfh-xmssmt } }

Param-dswa-sfh-xmssmt ::= OCTET STRING (SIZE (4)) -- xmssmt-id

alg-dswa-sfh-lms ALGORITHM ::= {
  PARMS         Param-dswa-sfh-lms
  IDENTIFIED BY { id-dswa-sfh-lms } }

Param-dswa-sfh-lms ::= SEQUENCE {
  lms-id     OCTET STRING (SIZE (4)),
  lms-ots-id OCTET STRING (SIZE (4)) }

alg-dswa-sfh-hss ALGORITHM ::= {
  PARMS         Param-dswa-sfh-hss
  IDENTIFIED BY { id-dswa-sfh-hss } }

-- The SEQUENCE encodes LMS tree parameters in order from layer 0 to layer L-1
Param-dswa-sfh-hss ::= SEQUENCE SIZE (1..8) OF Param-dswa-sfh-lms

-- Copied from Rec. ITU-T X.509 | ISO/IEC 9594-8:

ALGORITHM ::= CLASS {
  &Type          OPTIONAL,
  &DynParms      OPTIONAL,
  &id            OBJECT IDENTIFIER UNIQUE }
WITH SYNTAX {
  [PARMS         &Type]
  [DYN-PARMS     &DynParms ]
  IDENTIFIED BY  &id }

AlgorithmIdentifier{ALGORITHM:SupportedAlgorithms} ::= SEQUENCE {
```

```
  algorithm       ALGORITHM.&id({SupportedAlgorithms}),
  parameters      ALGORITHM.&Type({SupportedAlgorithms}{@algorithm}) OPTIONAL }

SupportedAlgorithms ALGORITHM ::= {
  alg-dswa-sfh-xmss |
  alg-dswa-sfh-xmssmt |
  alg-dswa-sfh-lms |
  alg-dswa-sfh-hss, ... }

END -- DigitalSignatureWithAppendixHash
```

Annex B
(informative)

Relation to other standards

The parameters provided in this document are consistent with the parameters given in NIST SP 800-208[12] and include a subset of the parameters given in IRTF RFC-8391.[2] This document does not include the RFC-8391 parameters based on n = 64 and those based on SHAKE128.

This document is currently limited to the hash functions SHA2-256 and SHAKE256, which have been used to instantiate XMSS in the RFC-8391[2] and LMS in Reference [3]. At the time of publication of this document, these hash functions are the only instantiations which have been published for a sufficient amount of time to be considered mature.

Annex C
(informative)

Numerical examples

C.1 General

In this annex, the public key, message and signature are expressed in hexadecimal notation. The private key is given as a sequence in byte following the order defined in 5.3.3.1, 6.6.2 and 6.7.2. The private keys here are examples and shall never be used in practice.

C.2 XMSS

C.2.1 XMSS SHA2-256

Parameters: n=32, h=10, w=16

Private key:

```
00000200 00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F 20212223
24252627 28292A2B 2C2D2E2F 30313233 34353637 38393A3B 3C3D3E3F 9D898033 E37AF48E 6A116F8B
15651CC2 67734670 07AD1937 5D38C23C 690C3483 40414243 44454647 48494A4B 4C4D4E4F 50515253
54555657 58595A5B 5C5D5E5F
```

Public key:

```
00000001 9D898033 E37AF48E 6A116F8B 15651CC2 67734670 07AD1937 5D38C23C 690C3483 40414243
44454647 48494A4B 4C4D4E4F 50515253 54555657 58595A5B 5C5D5E5F
```

Message: `25`

Signature:

```
00000200 8FF0300B E485DEA7 E5AE2C56 080302FE BC91B413 18C504F8 95BAAB0B 068968F2 22E5DD2E
120538CB 72D11267 ECF55F62 897CB641 643310F0 39BE9757 C5FA9D80 F71884E3 2447BA71 B4FF1C50
AD4211C5 D1075BEC A1A9F757 30F91222 E4C89C6D 1E4E06D0 A222053A 607B34A5 420FD7FF 8A1336A4
DA238C23 42D210EE EE69C469 751A6E7B D52DFB25 D0C63273 35A873FB 4C4E98D8 185FDD47 9D7A6B1D
40E1E39A 47FF0DB1 4937F33B 3103915A 9D432FA6 4142BF30 E6F53540 19C3B3A3 55883940 6C67B113
988587F9 DDFAC790 22730B5F AA222914 4E0FE186 14A7AE52 DB820266 A0A1F8BE 7D0E80DF BC9B6DAE
74319DE3 7C5255CA 3A70F0DF A85BA1B3 94ACE609 9447C3D9 D14A9497 CF483D4C 4A87E20B D0279AC0
6C8F58AE E923D4C7 7081CCCF EAE74869 1F1BC9E8 8E3DEF45 A0FA7807 6A066DC5 B3F3E790 34D42891
F29DF451 C245036F B7AF6D76 0E7BD436 537BD6A7 D0B8903D 1C56399B 2A95CDBD 3C0BB251 12ED77A1
2BCAB58D ECDC0817 6A105910 CE145A9C 57227AB6 D595E50C 063422A3 F8503847 AC6C0101 7F535A4B
36F2FB6A FC04948D 626EC2F7 A92555B4 61359521 2751DFA4 40E11664 DD9D9DFC 44AFB661 D1A953A7
C3E22570 A629D499 42892BF2 B5F72C00 A9A1B166 56E6F7EF 9D5BD7D0 865EEC8A 745164FD D7542C8B
E4E7B71D 2CD79B79 94BCA264 B4F0F348 46A4E6E7 501B2649 EFC0C466 F590525A 45270876 E2E94000
D6918E7A BE489DD5 0CB299A9 7CF861B8 89887CF2 023DA55E 521927C4 7D40DE41 3593BD96 44435085
BE3FFD5E EA1DCE50 D996D3DA 8971B2EB DEA1ABCC 02352922 EB1C75A2 0C79C8D2 914B12FC 3300266E
B78E2378 C4887E0F D9ED8905 C46F6350 C2B3122E 19962EA2 3113DE6B CFE0D5B9 BBFE68C4 BD25A777
F249D2BE AFBAA7B9 81C5A1B9 84007208 D53A65A4 F709E281 FCDC5795 B94E3967 A30437CE D628744C
7B2EA492 FA729683 8A791E82 68363A11 9E3BB780 C7C7B130 070B2B37 5025EF74 85FCD628 4119D235
DABD8EC1 A7768260 17E2D6ED 4CECF209 0A44F394 CEA92902 AE21A83A 2F2775FA 4695671A 2DF09A07
80ECB4DF 7F41EDC5 04817E89 CC632FEC 6EA1716A 14B44BCF 4704BBBF F6A014A0 01660EAA 5DF779A7
A033B3FA 91748C55 30A176C7 898671E5 7F324254 FDA2A0B8 10DE218C 4AA13E08 D9ABE307 0B4A8151
A70F0208 8B591012 6701F0A8 B2475C1D 7711AB56 A79AFD9B DC09622C 95D144D9 DE7A02D5 06304764
FF9936F7 4D4D0FF6 E4E6C6ED E0DB7516 652E95CC 92368D45 360EBDB1 2199EAFB 216E1B7B 09C6D331
6E8552E1 69EAE48E FD5C33B1 17879DCF 7DBD3795 327CEAB2 8D8A7C05 BD12654B 0ABC3AA0 CB7390F2
BB5852D6 D4D13F61 B94CD06E 937B51F1 83844A4A 78D0B79A 9CCB5C87 AA022E13 154370C5 9DDB1E38
B3FE2FEF 96AE94E8 8B2E3C64 02E6F3BD 87B77101 48A658E9 6DA1FC38 BBC0E2A1 CD192E16 B44D97EA
4AF81435 03035F37 71345169 E2655326 0DF269F5 11FC1CCF C26B5944 BA2B1052 16BED0AC ED790D1D
1781BECE F3A39A7C F46A8D13 99E44170 B4D94888 4F6BF701 BB2FE7FC 9D357944 DFCEC17E 49817F3F
```

```
ED983D2C 2D05AC4E 08F41995 95AF5C93 131A59FC E8E4A283 434C58B5 33EAED10 963C8D2E 4970E83F
1DCE8C07 392F11F7 770E8BCC C7DA3E71 035AA44F A7F5F3A2 ACC0D772 78C62120 FCE201B1 C7174340
C83F9E4F 1FA1A24B 7798399F D665E347 A39E994E E84E11D7 05CA2017 2C368A12 E1078AAE F2A7A013
BEDDEBD4 77AF5391 0187CDA2 9F1E0BC1 B435C5BA C9B6D36E E627395F D515FEB8 E7467A4A B6A8D9D9
32605B7B 73FADF32 90554D5C 2B0C8F11 D9787D4E 1FE9038D B6A08481 330C9C91 26E7E79B 48EBD471
4615664D DFD9BD0E 5DD3FBB5 61506679 D3A636C0 AB4A66B4 107F0EF6 978173CE 07D1A0B9 626BA0EC
EE498E3D 7F773F35 9CD7959D B7578946 D6236177 6CA7E014 673DAB60 791E3F8B BD553166 1D16D6A9
D6DE7E9F 03931761 55179A27 0BF5EAB7 5B7D6931 16AC288B 13BC0ED9 A0959A84 31AAC33B 169C64A0
3EB2894E 209EC584 5FA1D0FA C0030483 298319DE BB6F3C77 5476B627 8008D9EF ED184446 09111677
FED5BEE0 23842414 DC049437 D5896A14 48F93FF5 30E2C73C 4EA085C2 33268D5D C4498A30 1FB554C8
2151D393 A5D7935A 6F2996DB CFDBAA8F BB34D4B9 83BCBD04 64A2B455 C85623BB E5BAEC5F 1F5471F3
1BA300C3 135640D9 969E028F 56F62297 FD537D07 4BE35D52 45546BB4 ABDAB540 85C364E1 19A2020A
C2B1EEBD 09CDDD3A 02257E5C 35A79C9C 988D7D6F 47C99E11 3845F0BE AE75E0D8 76E91D0D 6B5F6164
5CD52A6F 0F099797 50E3DAE7 11D7FD29 B525804E 9471BB25 8C748297 37001D91 4A5D260F 4BF2401D
DA0C10D5 A5D39E9D C558A855 728744CD 7BA02135 F46F7598 3D9FFB5C 3D74A44B 9278086F 0C1C1957
175225D3 EBD02C68 2CCA76D1 31584E21 6C2C9224 C9AC81B5 785ECD7F E17B3BF5 179819F7 F9ED0E6E
976A88FC 51DEF3A3 A33E8EEA FB9309D9 39D03A39 01A4F5C4 7A0CCA5A C0EF9937 5C002FB4 538EB451
ECA08C7D 7C537F1B EF90679E 25D2D701 5B38EE1E FA663C98 34ED6BDF 2D90849A 3571C0A2 5B36079F
FC389C2B 87797E98 26EAAF28 6E3B4F9F D9C8AB69 D6FF1B3E 5710B2ED 394B3296 253F78EB 9C3D49F3
1394A8FC 03B41608 CF1CAECA 6E3C5FEF AB82FD9B BA0148D7 FAD8DBCD D478326D 28952B6E 781DDCF6
690ACD70 D5EFC75E 694D6F4A 0496FCB7 76DCB752 14916D1F B6617F23 491C31C8 A189DC09 0054B0F4
6E674CC5 EC38B4E1 F9F1D8B2 33075BF0 4CE63776 1E5595E7 A1D7433D A32E4BCD E54C3B9F 569B40C1
598705CA AD5AEF78 AF0BFA00 DFC83824 84FA1AE8 4DBC8F65 2724724D F4A085B0 43704563 54F3EE86
384D39F2 303F336F 06C95182 5BC56A75 C1279DC8 50F099E1 995ED02A C172E439 70150987 F8C7F4C6
BDD18FD2 EAB27693 3388CD7F BA187CCE AFEE6E52 9CD3961B 5BFEBCC8 7CE11A70 8430C274 CED608D1
790B7D51 BD661E42 1CBA67FC 20C4AF6C EBF19A3A C98DCF46 741FE073 05850EAA 75103FB1 C8635007
58BC376D B06F292A 6E556C9A C2A1C613 47E7D883 5F132F05 9C3ABC25 868BD9B7 5D80B95E EF2F3156
67D7D6E2 6045CCF4 87780552 A24B35C5 62505F07 43F6F089 F880E35C CA8B7E9C B849670B 1CF06AA5
759C573A 0AEAAB6C A893460A 7EA15566 94C69514 24FA6B42 C516369C C9658B81 B325D8D4 86977A69
919F6741 6DD362EF EE904D96 049E70FA 95A0544A 5ED85A4E 57429CE0 884F7634 50D9EE23 7769DE8C
96B71AD7 9EA5CA55 9B196DBD B9F96B69 41D0D8A6 F5F1A884 FC6A8028 15DAE957 E40BCE0D 4C8BA500
41DA0B5D 510D3D2F 662DB2F6 353B3A1C F07B243C D34346BD AAFFE5B7 ED4B64F8 CA7BA289 5F339632
92AF7376 538D2831 998CBE8F 076D2231 CC3A5DAD 0DA36CE4 9EEC00E4 0CC3A340 A40FC275 AB1BEA0F
4E96E008 A40D36C8 A6BD049B F265F1B3 DF85686B 53C623B6 40E3175B CB84959B 6D1E4695 5E18BC5C
F27D5FE1 3F72589A 395E1EE0 1EB9983D 5CE3E04B
```

Parameters: n=24, h=10, w=16

Private key:

```
00000200 00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F 20212223
24252627 28292A2B 2C2D2E2F B8E84793 033B07D3 E37A24CD FF2B9636 D47E85B3 5E074EC1 30313233
34353637 38393A3B 3C3D3E3F 40414243 44454647
```

Public key:

```
0000000D B8E84793 033B07D3 E37A24CD FF2B9636 D47E85B3 5E074EC1 30313233 34353637 38393A3B
3C3D3E3F 40414243 44454647
```

Message: `25`

Signature:

```
00000200 6F827191 2413DCDD A00FAD02 E64DE5B1 FAD32103 9E6E2136 346E1913 F598FCFE 81246695
48E8228D 43FA606A 18369FFE C41D48A9 6A9AB2D7 5B826BC0 BE195FA9 7A5DDAE0 77940936 B3BA9764
F603994A F1F8E273 3D9427B9 D8542C83 2044A640 4EC1F178 50C23180 E3F7CA66 0468F81A F41A3261
08AFD4A3 77146640 BCC24914 C1A4A7F0 64468F99 C8205CB4 8B5ABED7 76360B13 2A007BB5 A500DA34
1E51D810 A745EF9C 6DA13502 0A404D4A 22D35836 DD9F08BB 2D5DAEEC B3D7915E 8D962C16 4ACFBFE3
A9DE062B D4C6CA2B 51B8F4E5 F80A9F75 F1FD0497 4777100D 2DD12571 67B8F648 318EEFB3 55513FB8
3CF3E478 7F256A01 269DFD63 EF55AF35 81721E6B D0C8C35B C7D26A93 8D3310E3 BED9D927 7283D73F
655D56F2 17B50CE8 287BB535 01E94530 350B6E84 29430461 F3685E7C 4690FFD8 DA0628D2 A94915BD
C1F7957B CEA44A08 908ADEF6 BA340A96 CAE084CD EA82A038 A2B1ADB8 FDB77273 63603D0A 5093ADA7
B8AF48DB 3274EB8E 00BFA8A7 DEDB916B E074F6D3 61CB1ABB BAD1B708 A8509F62 FAB11E0E 6C6335F7
33F4B69F EF529D73 8AA1EF38 B834C653 2372EB0A 96935BD7 43A00126 5B72FC18 34430D7D 4AC2706C
07CF7AAF 2A0E3525 8F4408FF EBE74336 57DB8129 CC627D13 A86E8318 C2B2E067 F8148326 6396A407
81CC0EE3 7ECD4168 9CE41F8E C641A364 1BC2FCF5 C1D20B9F B8F794C9 4079FCD9 A3B274E1 7C220396
1193C5C1 D81B60DB 018AD68D 7CC25003 D5D21DD5 708F51CF 7965A73A 148C1D8D 6842944E 10B2DB49
F5A50B09 FDB096A3 721D11E1 96AC2139 63969FE1 C7FEA31B DBBD26D9 58E0BC25 A301E172 1D44342E
AA0378CD CDFD6B2B 336A06C2 5A3B28D6 C12E33A3 B87E9564 02BEC553 F47BF0B7 5E096341 4CDEC38F
90015D36 D231B856 9EDA5D8A F00BB609 4908F208 8B75C43B 36851C1D FC811F04 A3C8EBE9 4C17E568
5EB01C27 5ADF9B46 29127D5A 41D74C69 BD8D8F65 07DB30B2 C79763B9 4500355D 246EA9A2 512E3985
```

```
4A8ED1FB A13B72B3 79678412 3C729AF5 404927CD 4F1799F1 B58970B2 3E07F00E DBFFE5CE E58D3660
0A0801D1 B3F468E4 E83D84A0 3FD87AB3 E18C1266 CEC592B1 4804BE65 F4AB9906 6F12E686 7518B349
B4416634 87DE6746 80383064 C52A5A87 48739780 6346AEED F766CFB7 853F1AF8 BC84177A 37C78495
224539C7 2F988B1C CC64E2FB C7863F5D 3CCC551C 922EEFFB 467DCF17 511D93CF 14A5C05B 814C1FA2
56BF899C 3D327B9F F6419BCF F3B646F4 8BE2A3CE 37D7A583 59FBFD26 1E921155 1FC5EEA6 F363A479
A4688236 25257C81 32B9FC77 8F030F8E B69A0DEF C81EA4EC D9F90BC3 128358D7 903A81DD 27102FDD
571E7ADB 58423A43 9EA24E8D 00E68493 7D0A40DE 9364EED2 69CF4C10 A8DEEC15 A4266F4C 100E09A4
998AA122 48B57543 71409967 E0927A69 D53C55C4 E76884EA AB2DC64B F67FA210 AF0C3990 59F4CFFB
F53D44FF 9856355E 1862C5E0 4BDE172F DE3222EC BEFDC033 CDD7B93F 3E82C6C8 B03FCBB7 1C03D57E
3017A022 851EA6D0 EAE07490 AB11FBE7 88565A08 9D8D24EE 364F8360 B23C747F F39E4279 0B2F3E05
96B323A3 FBDE832A 90587FC7 B9C2D469 BF4656BB 3FFF2CF3 154A126A F2062703 EB8D8F77 D21FC367
710781AE 1E6937C1 51CE2CD5 D14B311B 752A5510 606C7FC9 23C4633D 11DD8FAB 629680F3 58320986
AED6A2F5 FDB51841 F287E636 66A79640 2797FC71 4AE9B2E9 E87993FB 89B84AE7 696F338A 18C6F522
C57C085D 599CFD48 DCBB0B3B 43CA75AA 6C44E58E 33FB5B69 7105DF28 439EFEAE CB929935 0F10B1DA
63B82622 E4A2095A C2F9846B CFB0C8F5 5F4412F1 6E978284 1A881E56 B8AEA36C 8AD667A2 CDC9FEE9
ED05A306 048AC548 1215F6EE B4B63BA9 53318752 8DB51153 83B66FB7 ACC595F7 ED7A042F F07EA187
BF093EE6 E8C2EB2F E0A8868D F1AB6901 2C7E44BE 6E09CED1 82B5F3B9 B39D6EFA 567FC23A C8E301BD
52D6740D 637CCAED 4C9BDFF4 29B9DC55 6316FA6D 04B18728 C8BAB148 A6035E4A 158B7A66 93DF31E8
DE2AD560 F2ADD6E0 5D8A99DB 2CD5FE1D 90DB2BAD C9108174 98256609 1F50471D 69D00B7D 006CF153
CD378734 7C54E49F 6E5EEF46
```

C.2.2 XMSS SHAKE

Parameters: n=32, h=10, w=16

Private key:

```
00000200 00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F 20212223
24252627 28292A2B 2C2D2E2F 30313233 34353637 38393A3B 3C3D3E3F BA62BDC3 9AF136A6 3E66F19D
3CFCDA23 2CF5CF48 5AEC1E22 C35D739B DC511425 40414243 44454647 48494A4B 4C4D4E4F 50515253
54555657 58595A5B 5C5D5E5F
```

Public key:

```
00000010 BA62BDC3 9AF136A6 3E66F19D 3CFCDA23 2CF5CF48 5AEC1E22 C35D739B DC511425 40414243
44454647 48494A4B 4C4D4E4F 50515253 54555657 58595A5B 5C5D5E5F
```

Message: `25`

Signature:

```
00000200 F88607BA B61D5412 28CDC83C 1EC48749 3A2D6D05 31EC6E8C EFC56373 9E2D75F5 1A28FA58
3861267B 9FD945FA 5BD2CEF3 E8D47AF2 6170ADE9 AA77754C 7E900968 9C5C3FEF FF494DB8 94E54491
383EEACB FE83FD1B 18CCA27A 5ABB0AE8 D121757B 8BEF01D4 844AFC6A B5425B48 30AC854D 2D580B1F
EA71B99C 74B6C9CD 17418AA2 C2D55D68 FBFE3445 F26343C2 71EFD032 ECC0BAB8 3226EDA9 C0B618CB
DCEC1E8B AA6FAAD7 008D6D33 7EF99D17 C5705129 1EAD3189 B029702E E22D86E9 2FF420CF B840E9D1
DBDF0322 48376134 1596C98D 84087701 C227149E 834DE9C6 D7DBA924 F0C4FE64 9BD76B7B 3F3CEED3
09BE524B 71612157 349E3B18 78EF442C 07037690 4C0459C9 7CB812DB C7350E01 4463FF0F 4C8EFFF4
15B2B9FD DE09CBD6 5334F390 25B8FB41 468B2ED0 C8D660A1 5034AF80 12D332C1 56094403 367466F7
DD7184FA 050E2D0F D835A141 E9685742 85CF5CC1 BCB96401 D67B29F9 3774910A 7087C91C FC4FCAB2
4A9504DC 8C9A8D11 332907F9 913CD523 DADD6B02 DFFB71F1 1599AAF8 0E3668A6 9C0CC9AB BBA85897
3814D76D 328ECF15 4BFA959B 071B53D8 D852ACC7 1C296373 21F419CF 7BEE9336 3927AA75 01BB83BF
57BBA9FD 8D4F59E3 EA47ED45 32CB3279 088D07AA DA9DC433 5E420ABA 6F7D9E32 FED1BE7B FDC51A5D
420D79DC 32969336 F224A6E1 91E52E5D C063ECF0 BE5D3B69 42E2CDAE 882E4C06 0A9BD16B 07301C54
7755EBED FD73A231 D437C6A9 07405530 4C1B4CAF 39D8C8EE 557F976C AAFF8000 4600D7BD 8CFF86D1
C8F314F2 EB03BF69 97694C05 F7A61BC3 E2188436 7A7C045A 2E78B5E7 BCBC4C2B 8C3A2203 5CD1D258
B43A757D B619B9C3 7A6B9310 CBDBBD8F 6933500B 17221F62 A9ECBF0B 477EBCFB 5645713E D67FAFE2
F7D91E80 C06F661D C44BEC1E 7343619A 883C8A2B 4A087A92 D13036AD D5BAFF11 89BF04CD 8D699289
FE312F98 73CD2D0B 54D5D3C6 60BC7828 49519108 B0B16768 9BBE2F20 69AC7F47 0784CD90 E6882A86
429DF00B 2A5A8051 3F0E6E0F 3FCAB5C0 DF79E0A0 62CA2C8A D6BDF519 A1CA38AC B8AE6888 D2D350DA
C390727D 29F39FFD F8FD4896 4B7508AC A81F760F A1AB2BF2 91F310D4 9B28D382 AA827D1F B5FECD2A
EBC136D3 BC5B42CB 9F0C7385 AD556F8E BDA1E479 3896C4CE D1F7006F A72094CA 3228F88A AF078661
1900D6B3 A1CA10AC 86F48CAB 73CBBC76 1DC707F6 4DD04366 37DE4D2E 0BBD19AA 66E1E736 A04C66A0
894C781D 022E1C39 58A4BD84 D6927D78 2F61A34E 723285F6 6372B575 7B705E4D 0EDB31F8 6D674617
C81AD319 52FA4E9B E2CFEAC1 21BDDD22 CF691251 1B647D9C 2F9CC063 509E3359 6E749D2A 53B947F3
AF5A93A1 A6AB38CB 02FE8FE9 19767FA5 6A619410 D5DB1E99 891AA8F4 AE6DC9A1 A54B5E03 DC3FBB46
BC2ED413 C39D16A1 47B4514B 13570BF3 B887C38A E1052459 EB327F4B 79EE0B90 02795C24 5D07251B
AE6CD0E2 308B9DC4 308D6C8B AB4ECE1B F059D46E 6C187141 6E3047B8 55B3417D 23968794 83AF92A6
2D9679DE 677C4A46 92AC9229 D1F6A0A4 E167CD6A 78DD85F3 171EAE38 C727BF76 95270CB7 71FE6964
E5B28A6D A71965A2 902B8B2F 7C766C69 D135C814 58F811C7 B61387E8 EA2FFF25 AEEEB544 FFB6B8B2
```

```
DB51FC95 1A0F0DBD AABBD998 209276E3 D666ECC3 6A99E1C8 1E18F3CB 1E97E316 5665F8EC 78A5274B
CC3F0EC9 C074825C 001DC7EF 733C54D5 43226F9E 1CCC6B72 B1467F93 D9F3B99A 3EAA6F4F 48B1C659
CF646149 E59B1A4F D663735D BA5A59BF D87DD156 7FF49A63 037CBA61 99514B71 AF54A91D 8FC993A0
957BB730 CD276C69 85754924 C785B80F 02799C02 8271D0F5 654190BA F779BF97 7B517A58 69F84497
F1F67170 4930FB82 E6B751AB 559143D8 73653449 2612E6C4 C3ABEC0D EE3D20C1 E16D8B98 6E0DF0DF
2515AED5 553A6EF3 6FB4C8D5 A380C4B6 F48782CE D8A52BA7 F4E045F1 5B27505E 07405E5B 3B932669
19DF6FB3 E8E287A7 B879D11B 3BB692EB 665EF61A BA330D41 AAC1DC00 B9C1EC02 7A017CFD 8AB23D75
A05C19F3 2DB1C344 5CF4EAE7 345C3F5A C0D1FEB3 DA09986D A7325523 46A16A7C 0AEB6E33 64788399
FCB14253 00D12A42 A3FAD289 7AD78F4B FD868D5E 5FC8228D F225759A 0E93B125 E4C4FE9F C585E0E4
1C7E4990 F008CFC5 90E6EE8C E14255DC D38B8520 EC053969 01836AAA C7595685 B0E10A08 1A09DA00
7D497981 004DD570 DCC399A3 53D3B78E 20069F96 A83C6BCE 4B005F6A BB5FA89C 73D34AAD F3415D3D
E0A8C211 A30524B0 9D4E2C0F 69F6D9ED B09B89B9 DC168001 524F8DA4 502F7158 819C65EB 8B278DB6
68E26CE5 F7A2D4C2 F6F60448 81C601AE 2736C0BE 27C40383 0292C11A 7DEFA9CB 273DFE36 DE7F8BD6
2707120E 3DBBBD27 AD2D2D59 62D97158 B447B101 94A2F24B 72F90FC0 B89A5345 7E9035D4 6F08F7E8
7FCC6702 8053A1C4 A08E36B9 F7E57559 11A17732 89ADBB78 763C7F17 4237CC1C 86FBAC95 4A09A27B
35E1774B 19EDBF9C F6CC2D0F 526B1B0A 4C574C7E 59A1F09B 572FD622 31AE1A1E C26C5C71 8C778CEB
ED7FDB75 485FC1F9 6C748620 7BC41A06 B594314C FB931475 254EF140 C4BE8B97 D52A5ED4 D235FE00
5CDECEAA 7030566E 83038ABA 2A5504F4 3E06ACFC 47333B86 DB3E96F3 50BA0CE1 F84ADFC0 0C820FFA
B44C4611 A6F14536 A9301A9C 47C12F73 7AF763EB 00166808 5F9E0C12 D8AF89A4 DC9B8DE0 94E1C87A
899EE498 FFFFF3CF 735130F9 F3F7EE97 11308713 D52A8A82 146F917A 9019AAF0 9D4E00C0 4ECFAE1C
855830FF 437CBAD4 BA741572 C07C7AC3 C62879EA 43743CE5 F8810D49 67982B3B 3B85382B ACC1BE7A
658872B2 C4C33E44 BAC96F4B 36B3DEAB C67334DC 48D17FA7 38328EC3 A74366C6 C92D34C6 36A0641C
E7D135D5 2DA0BA6D FF0EBF28 0289371D 25A70081 0EBEB648 BBDB9D03 FC27F9EE 26148C42 0A34E89C
AF14A8BD CF74811B 2FB26C01 CBA30D2A 26A8CA54 7B85D219 AD7712A5 ACFCF7B4 AD75BAE2 C61315C4
B2F60A15 AD645572 9AA50D5C 1BA807E9 418A49A7 CECB0FFA 3C3FE762 CD8B6F78 11E77749 F8C868A6
F5B378E0 488F180B D44D1EB1 1956B138 B9315190 272B9C82 6FC8FE20 48F0D2AD 9FB01B2B CDD7B7B8
B0E140B5 031CA2CB C34CEDE9 3C333DB4 00FD0B9B 380CEDF0 13B83D8A F246704D B3BE062B A5855D4C
D04EB5FF E91A3C28 6E000D4C 4E291B6A CB79A569 913F49A6 1B5CB761 242F68A6 3336AC6E 8EE39B52
9B1C4B40 7D5324DE AD9183D3 A108D553 13897F20 9AE50166 BBF972DC DF1BF774 496790E0 6E696E48
EE234098 80007354 9B31C453 11B9A7C9 38880838 14CE5265 BED3BB10 24579B12 8369A697 83728D1B
33481F2D 60845A42 AC320E73 85F6706B 3AC935C6 5F27130D F142E67F 1FA9A8FF FB3F0CFB F26CEE4D
39E820D4 E6B3CC64 3F758D55 A2D70A69 AF8914AE 1329EB88 69C7B7B0 B5389509 51D5C8D2 65084C8E
2F59A3AB 30816E8D 0F510559 DBAD1D4F F957C967 15E96A63 B33FB1B3 B5033375 37242BEA C44731D3
61D54A7A 4688C4FF 700B96CD A352EA49 DFD8040E
```

Parameters: *n*=24, *h*=10, *w*=16

Private key:

```
00000200 00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F 20212223
24252627 28292A2B 2C2D2E2F BBF748C8 60784095 8C52DF9C DAA1F870 5DD8E4C8 7D3E54A8 30313233
34353637 38393A3B 3C3D3E3F 40414243 44454647
```

Public key:

```
00000013 BBF748C8 60784095 8C52DF9C DAA1F870 5DD8E4C8 7D3E54A8 30313233 34353637 38393A3B
3C3D3E3F 40414243 44454647
```

Message: `25`

Signature:

```
00000200 3D46B21B 7171809B E5D5813A 8CB167CB 8B011E74 DEE49B01 B4B2F7F1 6E14E7C5 76941A5B
FFEA03CB A244AFAF E32ED739 02944738 D20FBE68 C681458D EF50CD58 F6163421 CC46DA94 4AF75824
33F69BB1 9147FEF5 ECDEE27C 19238831 7C349780 0B51A0CD 66DDB6D9 08708A1A BC64B880 27566A9D
839F7196 440147BA 3AB616CE C3D23DC2 9FDB40AB 54C15C72 A54EE0BA 2EA1672B 228EB63B E8AA8A66
944171E9 5CB4D9B6 522E732C A846883D A675C369 FDECB0DB 06B8368E 36040CA0 08434512 1A4FD90F
9E3DA8BF C6D05768 97C8655E 551022A5 CD8049D1 49C08A96 59DE4B47 DC89890C 04060BCC D81CD578
85919056 752F1551 3D8B44C1 C2675FEA 75048908 19B1A058 33565FD1 7C175EEA 42A3348C B3A17683
D73594F4 7362AA2D DC3B2C2A 19940D54 FEEC53C5 D68CEC82 AE5D7F2E 7ED77142 7EAAB553 7C9D9422
F2FEC6CF E635737B E3E177A3 8116C5EE EFB211FE 4A216ED8 DB810892 8F807AC8 60FE5069 E943D5A9
EB05E800 1BCBAFEB CA809265 4AB854BB 55A8D765 2D83AFEF 1C1A226A D7B4D66E A15D06E2 B38DB52C
01303F66 8487992E BD0CDDE8 033857B2 AF3B5394 5E87E8A3 B6A96A40 E6AEBF56 5A55FBC8 F7D77975
AF52CDED F67E66A1 E2F72FA4 ADE57E1E 487FA652 2CB53ED3 FA8B3DA1 5D438EC4 5F706CED 7F33689E
37E10CFF 26A2C10A B6E4F6E2 6C24329E E2DF2559 AAD94DE2 F76E5992 C31DEB28 FC403E0C 83796C94
457A82B5 F2FB04B5 2E51FE06 88727B51 5C3EC5E3 52E348FB 564DA012 FE07CE9A 2A4E9AF4 F8C24B7B
34E3217E 991B7063 54286BC7 98467B07 B5C443FB 73918B46 5503046A B521415E DC8AD493 FA748520
99BD1801 5233CB92 9689A19E 2B701827 1D68AE37 E2F6D6C2 294C764B 6429B8EF C5D93BF3 46AA1232
3B7D5E9D A51CCAB1 92358C19 C97DB772 ADB4D429 D521B88B BE8B28B6 92FCF0C5 70F59704 6D12B3CC
BE216A82 9C0B5456 8B0073FE B9F67EB5 77CF51AA CC0AD620 90524FBD 9BE81161 87DFC669 DBDE5B97
C3C3A59B C59403AD 744C39E3 D0CE136B 639F020D 6C80DFF8 5E2BCEB0 0286D81F 9552D0E6 F9D980CC
```

```
27AE370C A0DDC9CF CF183D81 F8D8B977 4E0A1A6F A1D05F10 B8C67822 99BD7563 8BD7A2A6 CE89FF59
3A55A2FA A2484F81 3957204E 2C46A2B8 6EBA1467 D59FAAE2 7A674D1B DAD5307A 1D26769E 38845433
5532D853 406CC7AF F0DF4D83 1CF88DE5 DC832E21 7E1D269D B7BF0CC5 53538071 9A47AC07 AF8D72AA
D20A19DE F1B2E833 AFCCA7AB 3237704A A708065B 808E7026 739BC748 24CB8C88 D4EA7E9C 7845C7AC
D7C13F1A 237059AA E82CA915 785EA4D4 EFB00CD8 A5C95314 5FFD35C8 AECA1861 C42E6DFC B898D6EA
724EACA0 FC6CDD72 221A418C DCFC268E 4485BF26 FD3F11D1 A1D0C804 8CA4A2F1 5133AFBF A5AD47AE
EBECD3F0 039AB901 F9AAC5FB 5C7F72EA 9621C669 4B39B500 B18E6AB8 37F49328 E548F4F6 363B5EBA
1D300922 B0956F3C 2F482BD1 2D74672E B9FAFC84 75D46EA5 EE91D433 B8DF9BBE ACF3EF4F 13E4B790
BFFFFBF7 6EF2D2ED 2BA4AA93 3EED53BF FE11F705 7BF7A136 B887790D 5057AD04 23D6C6DC 7A8E6871
456C5EBA D9C5F5C6 76BC7B6F 8CE06A9D E1CB70E7 0702C6CD CAB7E23D EC7D1E4B E22DC791 AD8EDE1F
FF6E45F5 02825B87 46D88522 7A51C65F 3D45976E C5B8074B 696E8CA9 1C6230A3 B2B6D700 1DE648DC
E4C382D0 58E90546 29EF9D89 9446106B 55A3FE28 B7853FA7 D98F2BB0 2675C797 152C7B26 339FC088
004CC057 ECB1C6D5 52DD6956 CDC4BCF5 F800B2FD 91AC3A8B E657672F 51558F63 2751AE1F 040883FF
F04F5ABA CF3D010F 3A345FFC BC617B32 3ECDA2CD 72643D4D CF77C8C3 BD74631F CFFE5F12 01C2B07C
203569B7 7DF05ABA BEAC3415 F269663B 7004795F 5041D3AB CA13DAFD B74CA432 50E5AFEB 66651731
1267299D 4CB8AAC3 35584DF6 5AB5ABF2 04B46F7D B86B80BA 1C566620 94574EF8 0562526C CC5B1723
E49D83A2 ED4B7DF2 AD21A6BD E11A84D1 6C9B200E 4EDDB8F1 2891ACB3 45B2716D E786941C C566EC9F
23800001 70EBDB5E D8AF60EC 57177326 E5A5A923 F23D637C 6EB57ED3 D42CE069 1F5F724C 3E4D6A1A
AE341474 1CA370F5 B0F1FD22
```

C.3 XMSS-MT

C.3.1 XMSS-MT SHA2-256

Parameters: n=32, h=20, d=2, w=16

Private key:

```
08000000 01020304 05060708 090A0B0C 0D0E0F10 11121314 15161718 191A1B1C 1D1E1F20 21222324
25262728 292A2B2C 2D2E2F30 31323334 35363738 393A3B3C 3D3E3F67 0E0C8CCA 74EB544D 358FABCE
89839FC7 3A6B89D1 A4E7D56B 4A45FCE9 6B20BD40 41424344 45464748 494A4B4C 4D4E4F50 51525354
55565758 595A5B5C 5D5E5F
```

Public key:

```
00000001 670E0C8C CA74EB54 4D358FAB CE89839F C73A6B89 D1A4E7D5 6B4A45FC E96B20BD 40414243
44454647 48494A4B 4C4D4E4F 50515253 54555657 58595A5B 5C5D5E5F
```

Message: 25

Signature:

```
08000015 2CAFE9C8 71DC7677 DE8F9E9C 100A86D0 1D0D6F11 D971C66A 3AACFCDA 764F23AE ED8E2076
1893346B 645CA534 A6F84512 351D23FE CDF05EC5 464EC591 14B65D1E E6F1B591 CBD83100 6E77ABD0
BAEC6007 81559667 5C208FFD B0CFB245 2714451A 153C0743 CEA8AEB9 A5E6DB2D 7AB1A29B B58AA64A
59CA8912 9EA63B11 92E88862 141721B3 9F25A4EA 75AD518C 19C962FD FAFCCA5C 918D2C45 D8B5E403
431DA119 44C5E2A1 E11D81A8 DBEF3BF4 E60C3B67 549309B8 8744B271 F912401C 515C46B5 DAAF3914
6F593FA4 BF74C7C7 EC043399 A328056B F3C4DB58 CD261D24 2E114890 DA0E978E 12B9B534 39F2025F
BEE8242E 9F6246E5 BC950445 B487CE8D 509CBAD1 FA1C401F DA882BEF 05FC3DDC 6FA52546 112E9FFC
98B20743 B0A3CB69 358A884C 5BA1A5B4 3D4B11C4 E23785FD F5A3F35C 818A8DF4 0EA60904 CC22D769
715027D0 BD2E81A7 3C375D07 9D1AECD6 CAEAE55A 3F2575F5 793E9465 DF1329CA 1FD82B27 4E8A7E69
08A3AEC3 179AC5BE C058A119 19E5F8FA F6A7165F ABEC1561 F9E599E2 C4470E07 DAC66BFC 232D01B3
86BE65AD 9F1AF249 0A3EAB3E 568F49FD C500D5A5 D03457AE EB478707 58E3DAE4 65D48720 066B2D5B
A3FD832C FB5B637F 8042F226 19453C38 A568B508 2855490F 5D07D5B1 D64B39D1 59571C86 85245DE4
CC0BA7C1 1F130E71 ADB405FB 038AF557 20E456A0 287CA489 BDC9D0FE 66DC9726 07CF80F6 8C077B43
077E23AD 94B31506 3A5F38CE B0237B08 421C41C7 8F043EF1 D735BE60 90D0CC6A 3BB58C52 E8195DD6
8897E00E FFDC3EA4 13AD7E2E 840324D4 ACC3F9E1 1CF188AE 3E2C60BD 68F760B6 97C7C836 32CCD88A
944DAE74 EA450A75 32622EE3 D24D36A2 01A867AF DA52B913 EDAE845B C4F1D976 8A5C6ADF E841662A
FDE60812 521EF466 6F292608 22C6066F C9281345 D0370328 7EE94E48 9993CC17 9A72798C D44860F2
6B88CFF1 F8C1539E E1C2EBBD EA7E1CF3 14D0819B 63FAE704 B81E1A6F 34BE95C8 218C5F97 4A7B9761
9DBA576C CBC9D3CC FE0BC61D BAD4492C 2A26C174 29044DB1 1F48FE59 09D4EA6F 2E573AB2 64B67550
34F8D2D3 8F8EF52B B1201D9C 22AD945A 04B75EB8 40483892 FACB0C19 CAD40029 BC441CAA D0E9B2E0
93E50A75 9229CBD5 99BA8069 B55BEDFC 02506C20 CCBDA8D3 F08A310D 4CB20D1D 26FDC417 71D95989
CC02FCA3 08734977 4FE4CFC4 5B74B2D9 E775E691 13E1A399 A8BB5682 85AE64A6 2A9FC618 6CC7ED26
C85BCF1B 44389E06 DD006E51 CFBFE156 B86675D1 D5BC57E3 7D1A9958 BE90FBC9 E984FA35 37F065FE
7B8F9C7F 61B2E051 DC4D6E1B AC4F6CF9 0CAA48C1 B4BB9E44 13AB400E 20651090 52AB0BD1 CCFDCC6B
16965E0A 2817C4D0 BBD911A9 1C83B22B 3282C9A0 2C9ADA87 9C197E17 C9A9DCBD B6A08FC3 50A3C261
C0EFAB00 8A637E94 FBF1478D D2FF28EE 8DCF40BB C8D4222D F3042F7E C21E4A0B 675416EC ADED4765
15A201B1 283C288B FB09F68E D58198C7 D27CF525 EC12AD57 11C16863 64A2E008 791423C0 7A195BCF
```

F41213B1 A98F1EFD 3FE080B6 F6BE9B51 B037C8F2 0E45CD8C 1BB2251E 3C56F41E 4FD200CB 4A359662
42826D39 130237A1 0C3BE44E 8487F6EA 52AAA081 645FB369 4C9AE8B9 68DEA58C 6124AC7C 7194F62D
2B136298 333F798B CB72A275 996F1B68 E53EE237 138FD7EF 3EEDE271 F45FCD04 E72AABCE 8B390D80
3D59838F 2E4F7F3E 4E0277C4 9C7E7F8D EE5F6254 7F39D5E5 C7157B16 554934CD 458E47A3 D1A4637C
1714506D D6A9BA49 35B1619E BADFADD3 BD5C47E3 1D24BE4E 004FD685 AF2821FF FC59C016 07AA75CE
1CD1BAC2 28F84F9A DC6ED979 8F9A1514 B928B091 A27952D7 311DAE12 D9FA7D05 3E0FEA6A 443364FC
5C1B99AC CEC0E961 6CA6595D E9FCF265 CC8102DC 4425D2BF 29AF47D8 C95C9928 38443B89 2C43A9BD
E9D7B4B1 7E2C5DF8 4DD5391E 5FF2AA7E 08326A50 34953710 0F069A9E D3DDCBBF 09F46197 A762FB5E
ED647524 9A9EB4F8 A0E95AE9 7EE66814 63B7E9D8 25DF305D 6177B0F9 B77F3B64 03AB06C4 85F03564
6C91A8C1 E4744C0F 7CD8B6BB A9791C1D F560F6DD E2B90F35 23DA4F97 C882CC16 BE936983 DE369DD9
E16BB918 B96DE9DB 1F3B839F B3B918A7 55C0DC33 F2A7ACA6 46F842E2 CB0A07F5 A8DCFC8F CB313E32
E4304BE9 9437E040 919349D4 2C618BC0 05DBD72D 841EB20F 556C7DAB 4556DFCA 47BE3317 8841D57D
2A85CC5A EFE22DD5 1526BFA5 A10DE334 82875EDA 1BB9EBE3 71A1BD86 CE7680DC 3302785A 182EBC19
107924B6 A4F5A4A8 7F133F39 9B497899 4DC9AF3A 3D220A4A C0930ABA 0F95F481 2D2CC2B5 3DBFC020
A9F411BD 30FC736B 87C7BF4F 1A383CD6 F556BF4F C8E7905A B78D5711 642FBC35 A570392B 3101BB89
5E8647E8 DEAF9ED8 7DC6D5B6 4431F172 55B0F458 D46CD251 8D2D8A14 B086955F DBCC98DA 251C056F
1A7A7F51 A29B4678 5ED85937 95EAC48A 97011B0F DF91282C D2CA9D28 E8F7B647 EE9B8E56 A4985CDA
94752DE8 1609D108 CCDD4D11 C8502AF8 DF478D3F F29DCD23 F87A3D70 A86A7D35 B48AEDB2 FA0FCCA6
3004D735 1C1C7DFC C6ADE908 EF24AA73 1F18F0EE 805A27BA A19BA0F4 4AAF9A33 85954432 39A59A9A
2E600F23 2A2CCDB1 A72DA799 D101A093 13181951 EB7635A3 B5A4B12B E0FC20C8 67CF3A42 56D22424
EA5A6545 ABCBDA0D EA610CF2 BE2B19B3 50F1D421 003A8958 BC5A3F39 C63B3F9E FF67F9BD AD5057B5
35263A38 B24C9ACC 9715B4CA 60B6036E 0904538D 3C98B66F 7C3ACE06 29A12B0A 0AC5BCA9 D31AB7AE
34975690 18A9E934 2B3C0889 A27DB957 0BF2838A 60472731 78EE0FAE B7A0F101 6297A108 5004CEAF
46AAC66D AE178156 F3ED19C5 34FD176C FA4CC4E7 AF17BA55 729844E5 48D453F5 45A6A6F1 C2FC176E
B113848E 724E835B 22A12207 8271B10A 3B0009BA 4C21A2D8 A6050F31 89DBF158 413C3439 E4D3A12A
9BAB9E63 AEB4013C 81F7E88B 6D2D1CE4 85F408D0 757B1DA6 CA636125 4900E50C 97466382 8ECB40A9
5C70F678 07774541 DBB1D2B9 03ADA6F2 4E4BAE85 FC76D3FD 646BDAF6 3B6D4896 B9E01ABC CC73AC73
3BE4E43A 7971BC5B 647A46C6 A698F236 3F481842 019D244C A6FAFA18 7D76A88C 5128FB00 13299C97
4DB7FB8D 504BC1D0 FB0EC5AA 4A969E11 32E7D33A E5444FB5 6C631EA0 C81D9E1B 0B3E623C 15A60A25
0E124778 1FB20778 9BF65E4E 2BAB3721 FF5400EF 7C6F709B 229A2413 23018C41 36C79248 9D9D7CF1
387881A8 F72AD4E0 F1365FB1 37E419AF 9F9695D3 6497BFFE 23166152 16000359 4E11D5B6 9B16AB14
DE5D4CFC DC338CB3 657EA608 6A3C0EF1 9C9C6B05 775DE583 9FB2E113 3EDFE72A 35ACB504 3837D7D3
D4DCA9DC 755D482A 9B2CC4C9 22250B03 769A5655 2A570EEC 2D53A001 87F0A898 95FA1969 3433BB8F
DD01344C 1EB6FB75 073385A4 B4188BB6 D1ADB7FE 51A3620F CD07BF77 F04E476F D3F836B3 4993E7A5
D48F0686 11C3EC18 C6A91B18 0B75D0FD 46663930 BB0372B3 EF3968C3 D07B0A0E 5EC1D6B9 49EE64F5
4C29B892 5F5F8694 3C581642 807D3A0D 8E911754 A9AB33C0 4B33584C 9C88290A 66507A6F BE60F0CF
1F35639D 27E76D7F F2EB80E2 23D66690 D1F1E397 4F026E2F E7BA5A85 68C4ADB8 0029BAFF 5D27C9A2
B5672099 E9675D48 266C9E80 2C5DA07B 5A2287C6 1ACF7005 49DE96D8 E9A26CBF A6135981 A00B90F4
407C1F1A A658663C 3F393639 7C32B5E0 7CD9F21A 752EC3D1 051301AA FB7E9FC7 014CDA74 56E96AC4
1DA24BEE E3B59E06 A998849F 23700D87 F54DA9AB 887160D6 912225A5 6D8D4472 9822760C E0EDA1D9
1D0F8F53 BBE951F9 AC38A69E 28876932 A7D4D2E4 FE89D5A3 AF92E69D A8763EDD D8630F46 E301A0E6
BDCC4BF0 E1A62CFE 2759BA36 98391D23 A9C5BA65 EBA84B9D F0E0D049 3357D736 C56BF43A 7A8E836D
6CF520BB 8B9EFB91 0D0DE40E DC7BC1C2 00BBA6D4 54E1A775 91C0290F 283A752E 49E5E05E 53CE1EEC
E01C719A ED97003B 4A1B449E 45351F0A 787F1CA6 0B0FE088 3FF17265 944ECB98 D3E3B3BD FF8EA7EF
2D8216C2 D3A10F17 EA7F40DD 150A8D2A C662A160 5D4F46D4 52BA6D53 060B8E1F 9C6CB730 492EC5B0
1268DE04 8AC21A2E 155C7475 F6B6B7A0 8DB3CC2F 924FF706 E98A961A ABD74664 4AE3AEC6 E8F74771
784E06FF E4A77502 E084DAB1 A5331CA7 E8728E83 BDF8A56B 2A8540A9 C202D2C2 5B4FCBA2 4523EB6C
5CB5610B 381D76F8 4639181E EC7D572D 3E40274C AC081449 3C68EAAE E7E1D822 6C91A2A3 BCC3E945
AD873FEC 72CA181C 4EB6ECB4 CCEF5160 BD766847 BCA0B99F 9EF677B3 A68B0770 90B44575 4A86A27D
E8373FCB 33FEB184 7C9141A3 5A59E51A 0E0DEE01 D0000AA4 C64B998D 16BE26E4 5EAAA57C 64889FD8
4FF4DC7A 6F3CA582 97B4642D A158A8FF 8BF05C3C 9C18EAF7 28A52F1D 2D36E276 327FC0CF BEEC153E
519EEB7C 8A822DDA C34AE774 CB506B45 AC93A527 E00CEA5F 40665557 AC049B63 52A3882D 0B3125C1
16702B27 9DFDCB33 7ADA2811 3C2C46C9 0CCEC0CD 8482D30E C63064E1 0CE60034 E321783A 5253F0EC
FBEABD85 0B3BF208 F497A7D5 66C79541 68AA953F AAB500D1 A3728AF9 8BAB38D2 AAAE4E9C 2FFC9FF7
C91DD2D9 E3907D69 8569C2AE F95D8957 B25361AB CDB3DB44 616AA6FD C4140034 4A6B70E4 4EA0E1EE
2F9ABC20 044BBB67 60F3FC62 0EB81A41 30EEA90A 651A939C E4AFA585 425E8A51 3BB029A6 AB0D0BCE
4BAA4191 2D87C70B 0475FA46 7735DAB4 42A77BC1 420DC96F A192C13E 409501F5 B5A683A2 6F63E35E
B06EA4B5 0A5995FC D000C99D 25DE54AD F6071902 73F1102F 8D98AFB0 1451D542 C8F4AB0F 12AB8692
5A2B69D0 A7757E74 1C5B82E7 628A1DE7 824DCCC6 0C7D6EB8 A6A4780B 24ECF84A 19C933C0 19533611
39B150FA 2ED01E1E 0CB3498F D3539813 9E5A02E7 1EF7EB38 5779D626 735F87E0 2FA77923 CEE8CA5D
704063E9 24583096 55E976A1 BBFF0DF2 6AE8E84A 68C6E432 087FE0D5 6254C661 CDF34887 6A411599
5B971898 877A20DE 8C596370 C3CEE930 3E190958 87B78982 3C2FE7CF A5AC00F9 D453317E B81F669F
0C6CD281 FF270E29 EC5D82D7 17E4975D B678F01C D916244E 58B2FF14 42062301 0B7E3DCC 9EF9CFEE
BA99ABE1 F6294ECD CBB54CFB FA52F349 D4373940 A75B4031 E6AA7FF7 00840238 2CA8CE68 C9EFC847
A8ECF0D4 7246B5B7 CD1A9FD3 5445CC14 7E93AC24 26A45B78 3E9207C0 E4E1EF5B 13A97A8C F9184877
8D4C01BF 0962E844 A1809D89 51433D9C 9169CB79 A538E4EB DE41AC02 19522857 8237560C FB48149D
59013200 63D8FCC5 FB5EC8F0 B843E5B9 0E52011A BE41858C 46963E45 B049F1CA F822CCA4 9A5AD02D
C79CB6A1 CBC8FA9A 0FB5F6BF AAFCFCAE 60FD29CE 963165DE F5F86E3E 30095B21 93A44C68 88F03DB6
61476FE3 2C6A8A4E B57B81AF 1C4F22C2 BA6494FD 478DBEAB 51D84C3F F0533A47 6DCCDDA8 51CF7C6D
3D2108C5 927743A3 841360AD CEF86BDC 6A2D91BC C0DF5E62 C4919D94 0051B98A 43EBF989 92AD4332
67F7006E 2772CA1C 5A73C447 94221F2B 184BE4D0 F42373B0 81964058 A163BE81 E214C88B BD8472AF
6D5B46CF 50E83618 9C930D57 A5586037 16FBFB1B BDDCFA9D 1F50EB79 5B9A3BAE 43352D44 F9F2C936

```
0C26BD64 2E8034AA 2D44E533 287F12F0 EAAB6CA0 6A8B913A 289A51C9 5FDA1038 5DF2B71F BA42F770
BD4CEAF3 D00CD9DF F664FC6D 9D2E73AB DDDDB283 C72FF70E 3CB84E97 93B7FA8E 08E277C1 240F92DE
2B6992F6 C6FBA936 1ED4F61D 30AB1FE1 BF349214 BA33CF97 0B3B4069 9A57E432 7276BABF F904F43A
7054C96B 1776149A B284125C 8DE59866 AAE6B626 8918013F B6E800DD 8E129765 9283B027 F0693F28
A9A4313E 93E5CF87 06C098C6 D12B2037 A6580B88 CEA75B30 F23223E5 21128A18 2CE0EC86 7F3128C0
44A949EF 7D8402C2 0079173A DA143EF2 5506BAEA E2BD8439 AED87B69 53B6E770 3FAA9BB3 6C7ABB9A
087F0456 C40F5850 AB6A4851 278C013D 34C74338 26239121 F12342FB BE667F78 3C973580 AC4BEDD1
39D7D2C6 C883B936 CC051853 49D41BCD BECC8857 B3E70B21 6F3324A8 22863F82 DE5D1517 FBBDFD7A
76975C5A 51D63005 AB60AB36 F2394BE8 DA36A9DB 06C9A54B 43649AA0 4C08523B F0EA2C58 BB464DB0
EEB0F234 7EC1A4FD 1A9B435B 77A0B5DC B8BB2FFA 9F1440B2 C1A0BAF5 0AD7EDB0 6DFF521D D9B6CA26
9BE7424A D860A131 02E69F87 0CE7380C EBAB9707 D3902B13 7A344D5E 2361E4D4 049373D7 6AE03BD2
1E4F16CB 05AA388E CED28221 886FECB4 6D3C8745 62C1ED67 094970B6 EF174A3C 47220A0B 30A85F0C
1C819609 C9D70D11 BD69E7E8 4FE5F0C0 56BB0B29 34052DD7 AA686B21 7DC6F3DF 18176A5D 85C20F2F
AC31FA39 B55C9F24 84F45626 3E5ED766 8EE0AA37 6527BF20 3313DABD D1883264 22F0E860 B9E99CFF
BFFC650E 846BB60B 66014448 CE3928AE 5D5760D3 A747AC88 16FD4FCA CF49D972 4D18D4A8 3480D73C
9924600B BA8E672B E52827CE 22FAE99E 6372B3E3 6E3A8AE1 AD5FEC45 3DB12128 1E52DDED F9E33B1B
5FC3BCAE 70B9532B 644573C4 FAAB0538 43E46FD5 7EB8680C 3F6B8A4C B39AC740 B732B02E E2FE4DB3
ED6E4C71 1C936BC9 5D59ED70 7978AF19 F6E7D966 D95B1B46 7F36E3FB C38AB2AC 7E725CA7 3BE39569
34267BBC E972DE4B FF64CAA2 103C689E 69C3D283 BC2982CD C9AF09CE 314358B9 9BB43C7F 1DD14AE0
94213D1E 22CB6869 DAF4E01C 8E2F8365 3708E610 24DF917F A3C40868 69EA9811 B49810B4 9FC033EF
BF9D742B 1CEC7811 744A2F85 6DE764C5 54AE81B6 0C7629C6 9D7F5070 D7272779 B93A1B4B 4DFB3B2E
68425048 F4867AEA 4BCD145C CCAD9C7C A0D510EA A1EF12EB 1F34E430 85FB0800 B0CE5575 81E24B41
4B66F8B0 7A261F58 C7A03DF0 B7EB9726 728BBC8F 9A1DAA03 EA22240A BB5D2266 6C5AF614 6860D191
DBA8B96E DBC1E4FC 9FE04781 72E1BF9C 6DD3A248 598BA7AE FB049751 9D87958C AF927F3A 645C4987
9470ABD6 0529D000 0066F523 836A03AB 6D5CB4A5 A82D3CE9 019F88A3 C929E77E 037061BF 66D95CE1
21B27E
```

Parameters: n=24, h=20, d=2, w=16

Private key:

```
08000000 01020304 05060708 090A0B0C 0D0E0F10 11121314 15161718 191A1B1C 1D1E1F20 21222324
25262728 292A2B2C 2D2E2F82 D4D48D76 4921D438 550FA2CB EA616EF0 B5B8AB92 0ED62C30 31323334
35363738 393A3B3C 3D3E3F40 41424344 454647
```

Public key:

```
00000021 82D4D48D 764921D4 38550FA2 CBEA616E F0B5B8AB 920ED62C 30313233 34353637 38393A3B
3C3D3E3F 40414243 44454647
```

Message: `25`

Signature:

```
08000014 6AE53C90 54A5E72A 5FF1E907 23355936 31D848DE 9CFC29F5 B1E33AC1 796F5056 6F247A84
F6A7A4E6 9E389E1E 0DCB50CF D76A34E8 BF779341 E1DDB96F F1337362 2F0063E2 3AA0D734 E589E680
48864624 A2EF835C 5DD48413 6DAEAC82 FDFADC0F 56F35868 B86FE483 33699874 30677BBC 031FE09C
943AA6FF 59666AC8 50C43997 2BE95C02 980E98D3 A9FF73D1 DD91414C BB566E92 CB00D6B1 D115F61F
58700817 F92C21BE C9A51D4B 58CC9713 9EA6E42E 176EA7BE DFCF61B0 7F0ED365 BCC6188F 1C469E28
69F9E90F 6BFBD002 80E8736C 9064B70B 9AF5CD56 D2C58875 FA3199A0 01735D50 545A40CB 38439996
6CC549BF C270D446 446A95C6 2DD04FE5 088CC802 C0D13DCA C1A8B209 A7C8C279 950DB862 5EB8E80B
B6BC17E9 50510A1E 5315BF97 267C633E AE94B48E F265A41A D5D8B49B 66FC69DC D4363388 47E484A8
C71135D7 8A69367A C6466538 04DF141A DE33419A EE2596A7 FC9F8D5B AF5973B9 0C470FFC AF5F1BA7
5039430D 79384D08 9E70C361 3DC415CB F38DDFCE 93649E0F DF9A733B 7AB2AC7C 448F4CDA 92E80779
7BD8E4F8 7789AED0 05C2FE41 1FC22782 8560C5C1 3AC08AC7 FFBD41C0 B93B4287 B51A1041 A651D632
5AF638C3 5248C459 87AD2AC8 3CF3A0FC BCC4F0D6 E18A28A6 2224E7DF B865B52B C5285F5A 6A4166E2
425B88E2 C5F5E58F 487977DB 8216A72C 74B184D7 E6B2F678 AACF7389 0E2FD40A 84506555 6BF91D96
E281A911 77CE2F59 EFB3BDED 85EB861D 9EF3688B 00AC534A 09CD964D 8F7FBD5F F3FB72F4 0E152789
BB9CB3AA CAE62809 C94B6926 910453ED 908C2A97 DED0535B D1DD5D49 77DC0C1A BB10010C 96385819
BA07D4C2 70CC36E7 60BC9BE9 3593EC0B F7EB1BB4 F5182C6C E2A1429E E0987979 E101D523 59185243
95C2AC4F 714AAF10 BDD738BD 166DA281 FE099A18 2A342E44 FA31360A 0FF45ABF 36005F38 45272ADF
9CD951D1 584ABF96 B831836B 5D93CFDF DB00B60B DBE4C0AD 648DB013 877E90FB 4C97EA80 5AD4C19D
2AA8A545 25D3EEF2 0512C20C 7C055104 4472C57A AD5C5607 8322100A 78DFADC3 2DC930B7 2BF7AA07
46B704EA 921106F0 B4ACCBB2 CC9BCADE 2BA90326 00C6EA93 415FD5C0 FEE28716 E6A9211B 0E8A2FE9
233BBE37 F4C4CD1E 403A2E4F 954EDD4F 2EA1C40D 75CA5E89 65695371 2602E548 F01BACFE 4E79FD43
B8F25393 F376F0AF DB87534D 981A7891 4EF39D94 DF82681A C28D79BA 56C21677 AC714432 17CE8B7F
9D741196 671AE71A 2E337778 A08C711F 7EADDA63 082A3761 DE700F6C 13DE0923 5A7D82D1 4D8172AB
BED836EA 53074560 480D83F0 49E80029 0CB48F97 7C68730F FFB9FC8F 110BE035 89419520 159DEC27
965C82D4 B51E0B38 FD40B475 62C0B559 B40D05BE D0EF7A41 F70A09DE 3CEE67D9 DC1B2C1F F5618310
04381FE3 529E4046 C0BA7EF1 7D54B356 AECBBD10 FF90136A A4CEC855 5EE467DA 12F0BACF C546AD1D
327387C7 5496997E 5E0423B5 F03C494F 5A9FCDC2 34385D14 71B831A2 F422D6DE 7AA09325 8995B4B6
```

```
9B2546B5 1293ED0B 6FF19D3E ABB94721 8E8EE391 F9FED4E1 1E7A7FE2 81EC6A51 63F069D8 B41325D0
64CE8113 414434D2 160D6A4F AC15B54A 8B09FF6A 9764B2E5 C3D1F58F 559FF5FE 35F253C6 FE708360
924FD211 A32C4D53 E93EA30F 0F41BB4F 680779D4 2A265456 A3E28C67 CBF32ACC DFCC80D6 3A9B590F
1E8E4607 933D365B D43D32E5 22D2F2DE E0E335AC 5CA7F2FE BFD40C13 B089F2C7 4B361F57 88DCB1D2
98068FFE 774336FC 3AF44B7F 4425D6E5 6FDB86FE 454C38DE 79949365 FE0FA9FD EB270D34 61806FD0
2B2B8CCE 78E81167 92852702 57601954 DA2BF4BF 7B91211E FD2DA2B5 AA0F611B F066FFF5 42C43B90
51C0AB4F 8CCBF929 7FAE29AF CAB4A838 94DF382C EF0C147A E3E4AA07 FA805210 05A4A761 4A06D98E
EC6BF3D4 6B50675C B0610C6A 41F7FFB7 1B6DD871 73EFFAE7 D6DCF4B6 713351FE 5FCAC6CD 86C4C5FD
A9B654C5 D0E62C42 3FAB07E8 9A5AF537 219425ED 9364DB70 68222814 FCDAC7B5 7AC0BE83 82706A73
D157AD0F 75C94219 128DE055 AB2B57A4 74E65393 201FD0BB 9E9901AA A24C1A74 3B854188 4B6EAFCC
9C798342 ABD969C8 8BB910C2 3CD9141E 0B4B14E8 C1B943CC B2ABFE29 7D1F1D97 0A86F357 50969D64
7CF14527 2C432FE2 40567C45 3AE3C3F5 41F98CD6 035D8D73 4D11AD56 79A2FB83 60A050BB 87344500
0CF68784 2796EFAA 559A7702 94EB4455 5A426C58 ADE0F8E9 49A985C7 9B79EDD9 E6DD1C2A 5ED64015
8AEFD39C 9A77E50C CA9F98DB 43C640F5 44F42E7F 4396026B 351A8E06 AA9C4822 68F15040 2B8A7E59
DDD63937 5F6DD11B EB26B710 72C9BD00 A8413914 91C8425E 4D654634 69771FBD 45223958 8BD970B1
72DFB5CD 2B6E55F3 B9364DE6 12BDCC28 524FCD42 9FBAF25A 3857B8DB D6C57413 152BA58A DF87D773
6BF27297 001610FA D3B17856 9DF0359C DDB1C99E D0204D9B 4DBC7979 E6DA8BC7 4E16A9DA 1476BBFB
EF57078A 1AB519CA AC8681C7 9ED8CDCE 87753758 4542A1FE 670D99D5 868F1A6E 2B0B4F1E 5DC8DAFD
3E6B8570 2CDBBA5A 5F3ABA21 D8691F18 F89699A6 9A07A0B4 263C3CFC 959ABB8F 270CEEB3 48ADDDF4
DD55EBE9 559EF4AF 01EEE060 FCB7AFFC D389AD6E 6D2D0F76 C28CB5C4 B3C250CE CFAD6DAB B2CFAEE2
61988DFA 66F8647E EB255DC0 239A18DD 2C366BB9 D1A7D9EC C32CEDD6 E77A38FF 970A6DC4 82F35E76
B8EDAF19 3F78A7B1 C2079BBB BE184401 D1023088 9D3FE987 A23E9972 45672531 6C51ECEE 89EAEDD7
862A1FB7 1C1C5131 7D80036D 75E2036E 491F39B1 FFB845D9 C6CB6EA2 583AF148 0316843F A92E4090
8741ADC1 06A6E01A EB96E392 1FAF012D AF70C795 7A78A3D2 A2028C60 52633054 472C7BD2 E2FC84D9
A3DFB670 68F878D2 CE650662 320F17EB 1E732C10 B2D5C7AF 029C9A3D 3161E73E 8B1AD7D9 938A1F64
9E0ADAAE 774DF639 70E1440B F30C63D2 D3BCF8E5 8F0C2DB8 7DFBC9E2 E2BBD6E5 61257C44 28231D1E
A40DF596 3C8A3573 E3A540D3 73E601CD 2C798E26 C42B92D2 C5507F64 F231B34A 64664D87 6C6D8865
47A09348 E14D7085 65D62445 69FA7330 680933A2 55EDD801 4CA77BF9 6C834021 0AF1779A 8CE4E211
B27234E8 F81F2ECF BF49E4E9 54A3DB5B 29015462 3808D501 3A64134E AA50C00F F71CF680 3C7DE363
8A300BE4 AE8B59FF EE19B1D1 F6DA4F86 83D6DCB0 D309F338 1E852BE6 385C7D54 24E1C612 533C1D89
A0B7DA39 EBB013F8 0552EE34 6881365A 00AB3A8C D6AAA5EF D6D82B38 BD53A4BF 92A3D6C5 8FBDB0CF
664557F3 1AAC7997 47992CBD 205ACB5A 30065D6F 127C025D 23CD4B69 142F3DCB 60ABB91E 4252B95E
C7A2C001 5E6F03C6 CD0D369C 06059D8B B4F95DA5 CBD0A25F F3AD3B79 0E5A6AAE D14F625F 7A2657B9
4DC7BBD3 18C93BC1 306CAB0E CF733A46 2A6CE349 CAB14917 BBF431F5 BA4BDF90 4B36C71C 89F10989
974098F0 78D3B791 11B024BD 8D6FDD2A 205ACA91 A6F793D2 84718C20 109A9015 AEEA36F1 860D1B7E
3783AA29 6FBDAD6F 64A81C3A 7625FD7B 041B8CDC 8DED0904 6FC32DCF A2B1739C 57F7611A B8B37915
289D1BC9 1FC779FD 6D990A3D 92F8B25E 72F7D879 8A51232B 0975A2ED 1D790FEA DBB97548 A84C96C9
F7526BD1 5E791CDD F6CEEB99 1FFBBE18 6F11967B 0CBD2D7A AAADC64A 3E64032E 44F1AF54 3A26B4D3
43F7CD9E 634D797E 3878FDE3 5D78C776 31D99DEE 8898B9EB CB881B20 6410CE23 A1974AFB BE1DB40D
DD7BDC5E 43E70140 D80A883F 5AA7CCC2 23535AEE D6CBA551 31505CD0 24C60E43 870CD3A0 82E38BD9
DC099F5A C064F343 95CFC58F C824D92B 7725D246 76E0BAF7 EDC68A8F D40E3B7A 8F5D9E5E 4FF9E044
83044792 EDD153F6 0DE8AFF9 320774E1 BCCD9EC6 3FF0FFD6 9B9A4B74 6AA99A25 62EED006 41B84D40
4BDE01A2 1766D01F 63168D23 6F8CCCB4 2D50E482 7D75F5E7 1E94DCB8 5AEC8ED3 7C59687D 72E14E9A
5B529AC4 23717CD5 51F476C7 D0960DAB 7EAD0AB7 6254DDB8 2177022C 9B4E8EF6 86267401 494514E8
3B04DD14 EDFBBC4F 8ED3498C 2DFCBA3C 59A068ED 01D9F3E6 7ED5EB45 36F9FE90 0FA4FF99 B474160B
1C3917BF 80AB9687 7E618C44 D152DF7D 8FFB95E6 24D279B9 FF7D5D08 277CC509 EED2D337 7FD80E26
D9368161 3490864D C6F02E1F 0E2A2165 87DF2502 846BD66B 91BBF6A0 EF75B755 AA9DFD
```

C.3.2 XMSS-MT SHAKE

Parameters: n=32, h=20, d=2, w=16

Private key:

```
08000000 01020304 05060708 090A0B0C 0D0E0F10 11121314 15161718 191A1B1C 1D1E1F20 21222324
25262728 292A2B2C 2D2E2F30 31323334 35363738 393A3B3C 3D3E3F69 C006C5D6 BE4B3829 4DE9EFAB
9E558303 8835804D 9F540A7C A9039F8C 40FEB440 41424344 45464748 494A4B4C 4D4E4F50 51525354
55565758 595A5B5C 5D5E5F
```

Public key:

```
00000029 69C006C5 D6BE4B38 294DE9EF AB9E5583 03883580 4D9F540A 7CA9039F 8C40FEB4 40414243
44454647 48494A4B 4C4D4E4F 50515253 54555657 58595A5B 5C5D5E5F
```

Message: `25`

Signature:

```
080000BA FCEBCB9E 55ECB1A3 4AE795C7 4BC20311 D62A6F4A 38D7EAAC 027EF5FD C974C334 C77F278C
87867906 B53EDF81 3EC6FDD7 38E6D37D 69721506 7FD28D85 5197508A 0A210998 96A5A857 494DA813
```

```
AAFA71B4 A3A686AE 187CC8D0 0086B38B 3223A70D 5E0530DF 3426DA9C 20AB14A5 813BDB8F E585621F
4C759C88 F208EDA9 261AD192 EB3C3287 1AAF02F1 A4BC55AA 50CA9BCC 8CED79F8 0F1AAC99 36B6F62E
25F44A5B 1D8C2299 ED54ECB9 74DA70AE 877A1280 8758C825 79F6B281 11EC96CE 05B99074 94B86548
9318008C 93BDF4AD B432FFDB 2CDB2D64 A6EBCBC7 047B9A48 2ACA6EF7 D8B07E6D B16EE422 5438B4D1
DA65DF09 0780C4C3 026FAED0 61E67214 35CAF6D9 57D20F40 F31BC1D2 DFDD6A6E 52D91FC9 E12FCE2F
B1E20827 9A904788 6E94D4CC DE57D3F1 C6085A02 23C49C0E 84E24F64 8409EC07 8267063A 1C027D46
E901BA45 D2A381F1 DB2BE087 386C3F66 21B0FAA7 D747FB3F BF1E5203 587E698F 8EAFE68D 2A2CD67C
95425F6C B460D525 6C92898E 7B77CEB5 A92B7E3F 82D6E93E 305D09D0 201E38DF FA3464C3 F3114EF1
BFD05917 055C30C7 C4A8CA4D 41702D49 49194819 8A02B89F BE08CA3D 7B22514B D4BA1314 383FC108
79179FA7 86B0F41B 9DF65A41 D23D054A 0410E168 68BD3193 2F81EA06 1A6645C3 96DF3CC1 F5C66712
309838E4 A405794F 8517D368 4B001F3C B68C2140 259BAAD7 354D1ADF E4547F8A D72ECDF0 762E3359
B9E0B0CF 11E45A4B 4EDA91E9 10DEC1CA 31853730 BBE84278 862D80B6 AED2FB5E 2B8D2BEB 9BD75884
BFDCADED 12039023 141C0741 0939EC8C 0B5B1F1A 67740B04 1E71D9F7 9356B928 1739CF86 F61BB125
5ECEDC64 9D0AE1E2 7BEE3DA3 BB7FAB83 26806C20 7C7B98FE E664DE80 445580E9 A94060D5 37F710F7
B8836A15 4124E3B7 77C87890 8E3FCBFE 79F7E75B CCC8B97B C6B83A4D 48477AE6 BF1A238E 075CD723
4530A02B 27204BE5 D266AB92 B6BE011E 24DFEA5A 3AEB59DE 0207DBC3 33A5EC4F 2CF79ED9 B93B4AD7
BD783E2E D1D2C122 6688B24A 8DAFF751 8BD95370 0F36D704 C043CC10 0B107C78 F707D79F 930DD6D9
85BB02AA 789368BC 4171FDB7 36349206 A7830219 135D4D90 488F2543 24069DF1 95F3C411 2F7C16C7
46316C7D 7454A845 28F9D54B 2C36BF1F 307BC46F C8A74DDA 13FBA700 349DA882 78413343 7ED87A0E
EDF5FE59 36264EDE 2671904D B50F4677 612773D8 641522F6 0E75999C E0D1A928 36B01239 76924F2E
D9FC5530 2CAB09BE DBAC0000 2DE41C76 0C2D8063 FD22120E A9BDE2B4 ECD1F565 2E844EBD A2CCC870
8C1209AB 6A97E504 68B9BC46 84DFA261 6008C752 C1038C0E 4584D56C 0041C7FA 5DA08E2B EC2F4C13
3626C7E3 67F96190 3227F0EA FA493248 31F3085F FCFB9726 AA85BA13 D29DE58C D8F9AA61 115E62F6
AA64F1AD A4DACD9D 05AAE072 493A3116 7E31102A E506ED7D 0F98000F B3E9ABFB A784D071 9D47AE8F
D9DF9517 4936EAF9 860F9F99 62004AFD A949B87B CD9B7A71 C47223FD 526ECA7F 21352659 659626B6
6E994633 615088F2 43448A92 DBC93BA3 D1657762 26AB136A D96FF4B1 D58F98E0 F0BAB297 21EA5559
50F87F2F D20D6C64 B963730A DA6B4285 2EB5A752 486F1384 91267801 4F62B48A DA0F070C F75E322B
1AE86832 C691333B 0FC52B0D 48E650DE 09DF562D 2390D716 6A1DFB8B D3A9E106 3ECBEC9E DA458AA7
8628B3DC 349160E8 B80C7305 D8CA9E5F B3AA8A75 08A8650A DB1F08AB C663AD6D 4887AE1C F8DC4445
264BA430 2119A2C2 0BE17B05 E4D87452 DB6FE8F7 18F22F2B BEF6F5DC 81539BCB 5F594F83 1F9FFCBC
AD91BCCC F87D1A2C B2558413 E17B121A 827337A3 98487F16 CA000739 A0D34BD4 B85FE5F9 03833458
A33A5E93 0671B9CF EF2DB2E0 23710A59 F66DAC7E 6B6CB287 32AFF3EA 9EB4833F B03E364F 25B083E0
4D9454C2 BEA3A7B4 AD9D2FC2 9FC5CCD3 FB9E5D17 AB33EE24 846560B7 400BA6AC 23E7551D A01FBED6
7D46CA25 106BA25C 340A84D0 10204CD8 778ABC79 07B0F93C F77AF682 4EA17FCF 75148346 B1B2E432
A90CE9B0 BCF09A48 3BCAECEF 171D8DA6 B610C58C 3754C991 F5B73441 7C6C094C 1B1064C6 AE499C0A
449C0F12 8BDBD3A2 C49A0362 263B20A2 BEF21C6E C616A7B0 B64DCD0B 60E53DE2 63EE4225 84E5FB00
7B17B2DB 9409733D E4C81DE4 357382C6 B97DF2BE 73BD98C5 B8611129 4C43077E 35463858 B781BF4A
B3735722 B4FAD59C D13F5777 486B346F B0DC08FF 5B8E737F B68D2BF9 77F2CBE5 7798BCB3 44F59208
188654D8 C7E025AA DEE89B54 4A7C89C0 EF001CB1 F87B68C1 C286DFA3 D92479B9 DCC153BE 5BE190DC
1DC09F89 16F35EC1 F6BF61A4 DA9AB71E FB502DE0 E85C360F E79C7AEC 53E76CD6 C97F8D69 84405423
D9579567 1698CFFC DCD85A35 7D806AB1 29D8342A A394F34D 72731BC5 15B7620C 4A3915EF 1251E2C8
1E91FF2B 450EC467 A920F594 2E8B1D2D 2E819935 B6DAE086 DC36CBFF 37051779 E43CDA29 3865E9C8
6510682A 65A37024 941BB568 24F488F6 CC7AB717 09E427D5 1083E88F 076B7149 93447526 75FE090A
221021EA F9DC099D 26663BEE 63251479 7FBAA3F3 726A2F26 7E58ED05 5BF82653 01228D56 949C581E
61CD2352 8D5AE4AF FA40703B C967F424 A65F0C35 49251D4B 941E5710 3299F1C0 9431E5ED 2EDDBAEE
F2DAB340 885B8A3E 1DFD2241 207660C0 F8F55785 51524081 58DF7E3C 889B11AA 5DC372EA 755E4843
DC9B5BBF 3E193221 3D218C0F 93F4A1D8 1561FBFE BAF504E1 50E12848 E10E6B47 358AC0E6 E4C494B6
A5C7EF0B DA2250A4 CC1BADE6 E9B120D4 8F0A65A8 1FAFA774 9BD34C81 86098457 4E2138C9 A0655D78
C00D5126 7ACFEC77 9D4A0B26 6538C84E 9C4E132D A32D791F 7C40E125 6BE9E5D8 D736414A 87BFDF25
96814642 103531AD F265FC7B 368C262B 14BE40A2 A7224E23 DA751A7E F0F77E7D 17344F51 FFF2784F
B92196BE 8931475A 38838CB5 DBD50201 0B88787B 4D4A290E 3B609C2E 44E3B1B9 B1031163 CEE7DE8D
36BA29A8 148C6A35 330B2C84 8930C015 5D1CE97B F4E1D485 EBDE1740 1518D2FB B0C2C46E 30BD2749
F184EA00 959E9348 72B0AC41 484C0E7F EFC4C8E0 234A1515 2EC868D1 D668FE4A 15C7DAAA B2AF1707
FBD000D4 8139BAB4 EDBC1D34 9948F394 F014B5B9 BC1EE7DF 0A62928F 63BB6DC7 64417DC8 E1B95C8A
29D863F0 E00B0904 BB917EED 51CE5907 69A2AE17 DBFDD0EE F5CF6EBA 5B3A14C4 B49E8BB3 C0428577
215EF30D 9B967842 141BBEB7 7A1A87F1 AF943BA7 55BDE7A7 22BCB6FF 4C5AFC28 3A31A0DF 3BDDBEDC
C3BAFE73 E56CD0F7 3D1911E1 A161B2DC 22424CA1 62BAD237 B39D8A25 C5AC891D 7CA4B3E7 77EA71B3
ECC24022 7554CBE6 6CACDF05 769A371E D69A91BE E6422617 0823E338 91FE63A2 D0844876 C1480CAF
5CBB54AF 40F4A149 752DFA21 11894C0F 6727D387 0AB8762F 222A9EDC 7B753C20 BA2E1D7B 7C93434E
4AE789FA 68A4C39D C826432E 7189B412 0BABE2BA FD385EFF A181F86A 97B62AC0 B2794432 CB90D12E
A0FB34E1 327CDAC1 97AAD381 87DEB5EC 28855105 64C50341 A6C4321E FE837346 4E4DDEEE C80D9582
6328B10D 2007562C 7D252B46 93E9551A 81418DBC C54C8272 B7BB3FE9 630F18D5 8687DBAB D2A491F9
B455EEBD 9F07B944 429117FC 199204E2 55D6349F 658ABABB F7F54E0C F5976602 36EE7669 2702C995
5AC4028D F0513435 CA0A099A 9C93D3DF ED7AFAF4 2A09C841 2EAB584F E681BAD4 302F84E3 ECCD0C6B
D7704DC4 23D68A04 18EFB992 90038B32 3E0D0E1E 7A165047 F0F4ABBF A3A592CB 44BCD36D 0EE67173
F6BE8D4D 8D9528B7 575F5D24 E6AD6CFF CC855C7B 6221EE60 D12D307F 9A55C1EA FD067AD3 A3948FCC
7D7E70E2 9CB7A460 1C753937 675B6485 F9735E43 7FA02196 30B3A477 8BDAE669 BEA5C047 AD44D50E
7D49548B F116B40B 29D3F7AF 921C9B3D 96F4A03C FE608996 408A42C4 24B78A0E 18E7AD2D 29279CDA
61FC27E4 35D6C67A 9DE9623C D3669330 6FF59A73 CD572F6B 09486B7B 5CF07EC0 F14F6020 F1ACD80C
EEA9C664 F39003C6 6AEB480F 27A991CC A6C94880 EE15962D 0D2425D1 D3E6F050 32F0634E 64BA23FD
4F7B852F D604FC6C 2E566733 F83AC36F 7C3A3612 77BBB561 2301F4E2 1EEB25D6 A8948D43 125BE86A
1E10F382 51FCD911 789C84ED 54D0DC7C A55CEFA9 26795B08 889AE116 F8A918C9 553E29BE 91134F59
```

```
EB3D43AA CD5C695E B6669B66 1909CE21 0647DB3F 82426AA6 8D01D878 6D746DFE A407781F 0EAA60D4
B40A6428 ADB807C4 7244AECC 3494124E FF2B25D8 E05E2675 020699DB 9AD791A4 C0926A38 6FA47681
F509E718 87783F1F 5F04B7DA 063FEB1D 2B0F52B9 1600BFD2 A9D76F15 0E64F80A 7BB9F004 F2E0CDB1
830AB3D9 C0564FF1 D40825F1 60872B01 ED431107 C02AB6B8 C855202F 8B906E92 BEFA4593 AFC851ED
B69510AE 114126B3 CA720FA5 E95C4E0A C04B72B9 02D788B9 754D6AD6 36665BFD 2B65A281 2BB24598
AFD74582 45A7B2D6 A04C80CB 79FABAC4 F8DE0D42 CB924566 AF6196BA 1E63647E 8E939CD5 09C02FF7
C13D5D1C D0D1E60C 6B59E27D B5FCFD81 613E498B D181BF82 CACD67BF 4C0D426A 041FE8C4 CD22B3E0
5016D004 05F01E89 05998B04 D0456532 99EBA106 273FE1FB 478D26D5 C5882F7E 96BD7FD0 1EE12574
BF839EFB 5D1C462D 434C5EE3 205D0022 1D234622 441361D9 8A1EB135 904ED83F AC76126D 7CE9B2DB
F8969AF6 583E6606 F2CFE836 2278CCF4 A39197EE 275F9353 0394117B C78263C3 EC54CC91 820DA20C
0CE63227 A147FE6B 636C7ED8 0182789F 5B7C0E1F B8D34893 D40C20BF CA43DFA8 E372124C B5746B59
C06FBF76 790B2C39 204509CF BF24ECCC D51B6198 6C105063 D27EF1D3 36CC5AD0 C419778B 888D1700
5EA24112 FB424FF9 81540037 CDCBF473 2CAE28A6 6DAE6281 E459BBD4 D8A84E66 EB71E64D 17DD2E8C
25F0E946 D18EAF73 CD5ABF5D 01FD6147 1701A027 E92AC9F8 B37FD645 841BC387 5DB4D223 A5CAEF25
4689753C B5364E0D C6E43529 CC08600A 9499724A 7CA7DE75 B361AEA1 C9388D1C 2F77697D 2EAC38EE
0831AE8F 8C690CC2 60889C6B E7CE9F03 01E8F6B5 D83D42BC 0B5C682A 048D3BCC F1EBDED0 45BDD182
11838394 ECD3452B 138A7E78 E61052D1 28DA879B BD1C7068 4AE61CEE 1BD5E50D 6D8D3873 292A278A
BEDE58ED 4CF8DF39 8E9A6291 D976C122 E7B196DE BF744F74 2885A047 CE005556 1EF0CD9C 05AB5C64
07B1B3F5 0BE4795D 90277D65 F561143C 565FDEBE 4CF2D823 6B227804 651D9819 E26559DD B77213BE
DD9362D5 F0F381CF 8AC551C2 3513844A F7047C5B B9279E7D 87C96620 4597C47B 5213EACF 28C76311
8EF36973 E85DB3C5 992E3696 73975A21 5EB495A6 79ACB265 81A5C376 063C73F7 D6161686 F0E7A5F0
A43EA69B 12038723 0B778CC3 32F0B96D 4D914D7D 7E017617 A1D2AD1C 499D0861 B9AE27AA 341201A7
DA23FE4C 1C0D88F8 3B3F8184 0C15CA95 8F9C85CF A8DC8DA5 35508572 273C5488 AF73C033 A0B24CA0
2B5C07A2 E227BC66 0895A07C E12BF397 9049B045 61E455C8 30D7BBC9 FE461D25 1434DDBF 4D9CCCFC
8A1649DC FECA7C55 21F1B863 7DD13D99 79A2BF6A 0F98DCD6 5CA10CBA 37AEB0F2 E3EE27F4 3E2ACDA0
1421FE41 42AB9F64 04E0E06C 22F4CDDC EB59737A 7C8ECFFE 5AC81013 86C39D61 CE9CA93A 17BB1425
A95C38F3 2C71FF4B 9C79D7AE 79D30A55 87C8B352 7B9D99AA 986EDCF0 21C490F3 1C336150 1E4ABD2D
3310175F 4F166378 50F117BD E7EA827D 8B5CCDC4 7E6F8A7E 21525081 5E1D1DCB 67D752ED 865512BA
C32059B0 24D6B9AD 52AB4853 B388A8CA 91952849 E5DBB6F4 4AA23B64 EA641BD1 BBF70D24 A58210DF
2719DCE8 6A8F089F C7FA0047 79B9F433 A84C0129 BFEFECCE 09A3F549 D4D8FE93 29F9EC37 1C0E7DF7
AC70F4CA 0D49822B B5C4645D BB09F691 C9580A61 43C9DB98 DEFDB16B A1AB3F61 C8C497E6 1B336A89
554AEECF 97E3D7BA 9BDD8643 5D7D1692 B78BA81A C78895D6 58515C10 9CCDECC4 449F2297 25726857
9569E9F8 597D3ADC 5242B567 600FCD38 2AA10461 05888EB2 93D3EF1B C89E3B7A E8016679 03DFDB50
87750066 8120C140 06EFDCC8 76B07E72 F7530ED5 BAFC5170 366A866F 1997C945 17542C58 14CEBD57
F1321C3B 0710E953 28E567E1 3BBE9963 82866D21 E8102750 0E3B660E EADCF362 D83879DB 5646D6F2
CBC4D7C5 B1024DBB F9FCD730 5D63C238 EC6DB565 B50C708F DCDE59A6 FAC58814 CDA0321A 250F38A4
E05FFE3A 4E8F2176 B76AF11A 4F92B946 D576B69B 3EB2334F B5B37789 B816DEE6 B9259D18 4E3A946D
20035B39 60539C1F 5B28037E F132B8C2 6BF9DE91 F048ED07 A52BE3FC C951C7CB 758B32ED 01665CB1
D35182A5 DB650A23 A2F91292 170E85C8 E9ECEB73 61F27C07 FD912C81 5F6982F7 E0503BEB BB4CE601
E4D3234E 2E791041 D932CBB2 46D6671A 71C78AFE 471B9B23 12324087 8EBCA68A 0D210490 D976B762
5FA17B14 BDC691A2 55806C6C 06128D55 96CFA792 A83D4715 AF49279D 6F65CC57 79AAF859 80BC0CF2
D1C3385F 19E88D00 726E72F4 4F8F14DA C120C4B1 D8D24F5E C0EAD7D3 68CAB472 81DE72C6 F22F5B22
F9971F3D FD8F0F0F 25B704FB 9E7104F5 4F76AB1E 00C66EFB 4111C54A D0E08DC3 F91E084B 32C14BAD
2097E7E5 E6AF553B 3F7C8157 D0C15E49 42BA1B69 ED6AC836 0B0E6BBF 6C815B5D 555E394A 35F5C4E2
B8C6F501 85237288 4C192652 36B56D0A 47AF27DC 1479DF2B 59040E86 25816817 B039339E 8DA9E923
CA4B7176 7482918E A3830754 5240A14E D6A9CA3E 59DA4B52 4855CE13 54E3FF03 D478B96D 835362EE
E44736B6 F1D4D7AE 23C7E20A E5E14852 9B7AEAF3 5500CB5D 67638BDC AE105553 4978B8B5 578C2F60
FBCEF2BC B7A61266 CA8D8550 EFB78370 F7F77E50 56E857FB A6024D42 3DCFF359 22154A70 55A201CF
44F56E3D 11E4CCDB A5823FE2 20249B0C B240E2D4 42BCF33F ACEE9B64 3995FE0B 7BDBB386 58BA74D2
F37FD530 9706A5A3 BDB3F51C 16FC701B 278A80A5 F1DF2FC1 C5F80493 D2C6964A 33067392 CC62E3E8
C76873
```

Parameters: n=24, h=20, d=2, w=16

Private key:

```
08000000 01020304 05060708 090A0B0C 0D0E0F10 11121314 15161718 191A1B1C 1D1E1F20 21222324
25262728 292A2B2C 2D2E2F70 1B02B281 ACB7F8E0 82098C35 9E2BB987 3D7B3603 41F12D30 31323334
35363738 393A3B3C 3D3E3F40 41424344 454647
```

Public key:

```
00000031 701B02B2 81ACB7F8 E082098C 359E2BB9 873D7B36 0341F12D 30313233 34353637 38393A3B
3C3D3E3F 40414243 44454647
```

Message: `25`

Signature:

```
080000E0 199594BE AC866B26 FA250A5F DE4028EC 2BABEAB5 8CCBA382 11005D0D 265FABDC 95ECD4D7
479516CD B15DDC36 86079EC1 269B1427 3D3B282A 807FC7BC 4F84A367 887066CF 69C36CD1 5CF02AC1
```

6F45ED4D	EA2AD7E4	AE041FCF	6457F5C5	90CA0CD4	411E3E7C	CFC02E07	655F0F5C	583C5F11	F76A9D72
B560C366	F1A50E31	6962F708	5DD185A2	1A4864E2	0755AF3A	B914ABC7	C6DC9466	DA8556ED	747C5A37
1E25739D	F89C8E50	694150EA	8F7549E2	EC76514D	81517556	A03FC08F	9DEB5AF7	03B63C17	3AD4A255
6051649A	B2C29E99	A28B145B	F8F836C2	D3FEE321	47F057E8	AB937D06	087F9FE9	5D53410F	01553BE0
956923F2	B441AAFA	69BF5BE4	A2B60D2F	8881046E	9717E80F	54BD9F40	4B054588	85838342	A42CFBB9
13F9529D	DFBD2F49	BC3B708E	7A460EDF	5BE63260	9E848ACD	3EB11D03	C3B187CC	1D068E87	BC78B371
54FC1720	32AF7864	0AB6E13C	B8A190D8	9B3E90B1	948892D9	1767A908	0AAFAF56	DA54A56B	FF49C541
472FBBC2	9EB5208E	01EEDF52	B0D3C260	77D50A18	2A2D9934	0B651EBA	FFB69BD1	E6E51A9F	CE657D50
F69D179F	89E9F9CD	BFD33722	1F169F67	ED959424	F664A2D3	F7D79E5A	C81CC286	7792D65F	ABA667EF
2DAB0953	49624AE0	58A45020	FD8511E4	C51C63B9	2EAB23D0	F74153DD	7422400F	0DA21255	06F54C5E
0FE23094	1F686BBE	4A773543	9D6201A8	7E53EEE6	0CAA60B5	D3A938FD	7CA57749	A6C8FFB1	67F67646
71EC9356	692A1C26	CBFAEF2B	80F70445	1CB2F5B4	6FA91F62	ECDDA0F6	72161280	450CF528	66C92866
773386B6	848E9FB8	D3A82F54	CBF28647	60FDFC4F	9D6F8AE8	79CF3A9E	9A145B56	D9214161	47C172A8
C63343F6	D792AE65	CFE05D70	7F2CAE17	DB0C4F4C	99727887	4D20BE66	5C207662	0E33709C	ED166816
669D3549	446620B9	934DEF53	BE60EB71	EEB12633	2B22ADE8	44BF3218	5C28E802	35CF3A24	134999FD
B13CCBF1	8EF20697	9B036883	4A97B5C8	81C917D2	44B9415E	5EB6B8B0	A80FF9C8	F9390AAC	614BDA31
2D1FF59E	65F3608D	99140986	EDB9280A	14CD8086	FF6AFC4E	AE6291EF	8C2AD826	A2DAF43F	923911DF
3E8FB816	589ACFB4	AE2D66EA	6BFA6DEC	E7BDFEE5	1B364EC7	A9B84384	9B5A4825	79D98C68	B9DD2AE0
27114D3D	998D2BD0	89C5C8B5	612984D9	E1B58174	AE7EF149	7D7EE1E9	E7DAFE54	08A4B2CC	B510C304
15D60C22	4704F301	4ADDDDC8	76AA6D47	025C651C	8F5A5F6F	16A0361A	082E03CB	788AA0EC	391455C8
AE349900	F6D92CD5	BA459614	C1464BA7	45F102FA	8157A14B	07413D33	FB031832	82D590F6	8B8B8B6E
3D316E7B	83E9CF08	6CC51539	C6698725	8F43F1C4	DE024BEB	316DBC8A	3BF3AAF1	1EAAA614	6FFE4D0C
336294FA	D94C59A6	2DDB5636	D398208E	E26312F9	7E857CC7	D009D716	26EEA82B	8585B509	2F8084DC
1F2985DC	2253D0CD	56065C1D	BF45D93E	586AAD77	80A504DD	B3FFB374	837C8A8E	89C3A2F7	FE5F24B0
58CA26DC	A05C2470	3B57893C	8CE130DE	81B60CE1	F69520FC	559CFB37	850D8F2E	B9350F13	AE8B2193
DD80EEAF	5D9C385C	62B8418E	A5B76C48	588EDB82	5C004180	55AF3B7F	359F5F10	94288666	3018978E
1FB4EB07	680EAF22	8B646B01	C402AFB5	01D8E863	14E9401A	7DA46ED9	5ED0D180	D71F6668	4D206DD3
83C37DF8	83EF5888	0D7862AA	F29C9781	5335B79A	6F072BF5	F8E78F6A	C5B28185	FFD4AA10	E248B338
A1C452D6	0ECCF077	2AB9EADA	187B8F64	F3FDFAFF	F6707406	BAFE3779	CF3994FD	3DE4672C	7BB8BF69
1616EEDF	2D193C33	88EBBD73	65FDAA5F	64C09CDD	0DBEF9E7	00413350	04C7EB1A	9A5E1B0D	8B20D434
A9E93A68	8E0E7D36	BE6291B7	DDCA1221	2DB045B9	F67CA401	78014A34	D4924BAB	50544F2C	2F395189
A72B3081	C2F36743	412559E6	683AF7A7	7DFA78DE	65C121EB	052BDCEB	7511BEF9	3A843A22	11448915
962A6BE5	5AD4863E	71A31BEB	7C9BD705	6EE3C5DC	83D8517F	098CD819	9A2C9173	6CCBC3FD	436D7C3F
98EB3D74	F444703A	DC990245	B12F251D	9212B844	ECB8E2D0	09530A7F	BC7207F8	FFD1A773	C2A20C1D
3F8BCEEA	7ADA7DA1	7391B33A	FDF4CEA5	1E9308E1	0F4FF6D8	FD044110	ECD68668	6B3853CD	861E7CD5
2E50D7B6	58D19317	F22C074B	B458C140	67F12730	CC9346C6	B591C2BB	FE7F930F	211B4B10	81066C68
925DF020	459F08B6	DE96D0B8	06331D1D	542FBBE1	5A5FFE0A	F23675F9	B6000138	0A7A34C5	845D2A65
97B4DE34	82759CAB	25741882	AFFA4BC2	0BC224BE	92BE44E9	0585BA05	31E6DC09	D992D6EA	5D31D0CE
635E866D	6B2DE58C	D88A4719	8165ECC5	A225CF1F	3371B4D5	D77629F9	963AA408	C6B26641	5A7502ED
FD5F5F7E	23A35784	CBB4B9D2	2D6536FF	7359DA27	1A043C66	65E12D41	3CC0C7A5	B28EDB48	61DA0043
E95D1B8D	0ABC491A	0BEB8E7E	EEB98E9C	8F171984	BA4F16B1	B8B933BD	A54CCB31	A04F32A9	C9EB8C05
F9297F61	5AA6F83D	AE0E969E	9AD8D464	7BBDA27B	E6DC5E3D	AE538975	C8D58BE8	E82E6A18	E078EC42
8894D61D	3FD850C8	C460A3B8	57287C03	9C399E8D	A88333E8	B30FDA2A	B4433A84	316DC7D7	92ABDB15
9B90A40F	597544A6	92A87C48	B3FF994E	8A80138A	58DEC9C2	7515441A	DFE9C297	A3B49E84	7AEE0EF2
446D9B85	9129C429	8E254B43	CC0A5306	BD87C052	453EA374	0986BB91	0342267A	BFE0A91E	5BD89B4D
3A53551A	39320D25	F38F224F	7C23EB46	9BB5BD4C	36602DC3	554872C1	A1854569	8E152679	129BFE17
BF3B7D02	AC19465E	8824485A	36227756	6A2D50FB	6B27C2FC	777A47D0	7B68B32B	0E4AC86A	F10D1200
5434C31E	4C13920D	77DED774	85661006	04C5BC2E	5F808511	350E6862	DFFAED5A	E0B7E52A	15AE1C4D
993D556D	852CEEE8	95A24BF2	4DE82129	6FAF3566	0B27DC87	B4248119	7CAE4CF9	19841386	C35C78AA
718ADCF6	964F4170	1D26F6B8	45BBF884	D07A8C57	819E860C	D33E1DA7	3E75FC13	A993E3D6	8B475DC1
9626FEC2	43CC1ADB	59CF9DDC	9FD03A4A	85FC0AA2	ECA08D23	7B760789	9033B389	92763CED	B38AA0E0
C50B4218	E619D322	35FE099C	0A267CBF	051F7234	2C24F18A	8704291C	8CAF6BD5	A1E45D27	2C2DE6C8
DE565A19	1A0B24FA	6F7DB930	9B3E2C3E	5ECB2908	0734D2BA	170FCC57	38E4786D	372C5EE0	4E03D3ED
761BDBB4	D303B260	E97FC815	2B8E8DAF	F58B17DB	544D971C	3B89A48A	D095C9F1	3FD03131	CA0F1C2E
A06C1F72	82CE57D1	2C265788	F5FA28F4	1DF49F58	D7A671FC	14D1B9C0	0D109835	F8A905F0	C4073FDC
A25F9EDE	66048281	5AE91EC6	F974A7B7	46CD0583	11EBF5E9	51AE7F44	DCBD4617	DF2D99A1	6CF8F81D
6EE34163	98D74411	312E4C4D	3C7E742F	5F9EDEF9	CCA0EE31	A26FD81F	E4249FF5	A7A4E068	BC21191F
27DF11C8	4CA086BC	3C8CBC67	E7C4AD78	7806140B	6A16553D	D574262B	B47519E0	7EBE8D55	2628D2B9
E6018CEC	39A4780C	D5B06D27	54EFD2F6	F6A50EC4	E7A2F67E	809B89EC	8A55A2FB	6B31E411	F98327A1
3CC0A26F	68125549	5635810C	8889B279	E4D87B2D	DB3F59B2	0E1355B8	CAFDD00E	555C339D	44A3A396
41ACCF66	592B489F	66DF101D	A226A0C8	3B24D3C3	18B2F68E	774AFEEC	A20A009A	59E0EDBB	80E60EC4
8541AAF4	FF0D98FF	44C48BBE	21E55DFB	09BD7392	39B8CBDA	AEEB88BB	E5BF9A4F	5C81CDF7	33BA00A1
8D9331A3	6ABCC1F2	1051D70C	B4B1CA64	E7B420A4	380A8A0C	64C036F3	47893050	D669B0C7	F589164D
B6B1AD08	EC72AC8B	FC37B3C2	C2428D6E	6B15E40B	E50DEFD8	FE332634	7B342CB2	089F7421	F6869000
F45E9A57	A99C1E7B	50E0E5C7	DA68A5FD	2CE8407B	A01247D0	54B572F5	B384A72E	CBC5D17F	7EF53E60
B9D07C36	8F9051EC	B6400C1D	53B06E08	F1E1CCFF	89F67E2C	C5935796	52AFDC71	B6F78408	E363A59F
8E05D915	5672709A	D2A17F7B	4169BC17	4CA34D2B	79CA08C8	0FB83F08	4E046B45	F6DFB8AD	9A991B2F
D09D67C0	876EB802	8761F18C	2DCE3D6F	82AD23C2	42C5DF1D	A7AEA583	AEDEACF9	6CBF2695	629BE772
A6BCF539	51850069	81D1A679	9A906303	D16F05F5	6CC85845	EC86592A	BF44E77F	49655F24	F31D755B
5EF4E771	E2F478D6	72CFD352	518D8D69	E045BF5A	116138A5	4655BE6D	041FBFE3	F450C903	24A9F207
CBAAB6D7	8FAB2DA0	94A9E53D	22F86050	349F63C0	463A461A	D3C0F78F	843753CC	E5FA184B	D76325DC

```
6062A8B8 3F00BEC1 1051A32D 8D088811 70D9D4A5 60E7BDFA C99EDE0A A9AEFA18 C2D7F5
```

C.4 LMS

C.4.1 LMS SHA2-256

Parameters: n=32, h=10, W=4

Private key:

```
00000000 00000000 00000006 00000003 0F0E0D0C 0B0A0908 07060504 03020100 2F2E2D2C 2B2A2928
27262524 23222120 1F1E1D1C 1B1A1918 17161514 13121110
```

Public key:

```
00000006 00000003 0F0E0D0C 0B0A0908 07060504 03020100 A7CD5F57 42C84B1C B5790917 D10DA7FD
4435ED69 F8D5951E 3DEA3606 02F488AE
```

Message: `25`

Signature:

```
00000000 00000003 7AD25F69 6309A956 A00CFA31 53016A62 3B27DAC3 144795AC 75A82ACC 94FFB0B6
B11BF66C 58AB1C2C 13834027 64ECF73F CDF25D15 8A348209 B3E36C31 AA62FB72 42554201 91876726
AEF21CA4 B40F24FC 33DA7F8C 75951F57 14B44683 8530FE71 8BB32FBF 88223E97 A3244C26 BB4987D2
8F6A4AAD DA8BCE7E FDD92FAB 6CF52F96 BCEDAB04 EF031388 34CAC96F 81E8F2CB 43024176 A5D02E35
999A0580 66A2B625 A4D668B1 357F030A 4D9FFFD3 6BDF88D9 2402F348 37496C93 8028DB38 FBEA8B84
B64BCE63 D5A93C7A E09818BD 039D498B 7FF3A2C3 0E2ED8C3 9E35A2BF 48AE92CA C4058707 91E611B6
BAD9EEE2 0C7BEE9C 24A2AC3D 6386E1FA 3EE97EE6 C972DD8C 033C5112 87E4101E B8C8855E 78A49484
D2191F64 0769CE91 9B9DF12C 32FA0DB0 33ADB9AD B7FB8733 8391BA9A 1E713E99 D3F22F23 A864F159
3465053A B5B62BC0 08C9D5F7 2EF7CFB5 3570A41A 4D78FD6D 7F9A99C7 39C23809 5A47A6F4 0D6064B6
66B92591 21C822E9 E2227C13 59A50642 B9D2FF52 DFB72B0D 323A4F0A 75FC2155 472915B6 B98F1B0D
66D3CB17 C0767A3F DF36E59C 9FF920B1 7876136B 9F86FE3F D1C65947 59C26648 7F876022 F7038C05
CEB654C7 0117DE6B F1E90AB8 C17E60DC 9BA7FA76 CC408AA6 B9F8C320 624C83D0 6F5CC791 B38D00E5
0BF50C7A FAC87AFC 68C60EAE E9B55426 001B1B96 EB43A1F3 06D29D54 0A8DC212 F3CC58BC BB25881F
E54B1902 5A5C468D 24CAF765 D69943C8 9E3B8EFC F6CDF8BA A487F0A2 942CAFBD 0AA7E4E5 DEECBA7F
53245A8A 6ECE4FA4 FAB83656 15E31C11 12AA1952 ADFAC3FD 11AE3CAD D225C34C 0CE43724 BD4E46ED
7E52A6F2 2313CEB9 64FC6C3A 5B00A57E C1F9FC16 BCF67147 616E4623 107FB06C 10199009 3B1E17A0
D9E42B73 FB43A965 E7D7E827 057843B7 99EC64B7 5F3EC8AA ADE85576 5D8066E4 28F89F44 3D200B94
A686D24A 2CB60D23 3A37E0E0 316C52A2 2C9DEE01 A3A308F4 4B9E8BFF 213B7FF2 FAF63074 B892E29A
AC90B710 82C3938C F2C8ADBB C8E3D910 84DE9476 5E889F16 6B4438ED FDE40ACD EA2DC957 D301560D
04DF642A 1BA48AE4 D1AE305F 2C7F6121 A7E96561 46020AC8 16B3BC4F A4E0F373 9C4B441F 19437942
73A43675 D257CF2A C4649D03 4BA275A5 5A88FE55 5ADB8133 6E72645A D50AD95D C07F3840 FABE46A4
20A866DC F93B0D55 F6DCC2A7 6B0D4EC7 E5C330C8 99C682CD A69C11C7 A03C8B52 530757CF E90CDA82
B3E6814D 5AD0F497 C80596D1 F4506B67 D055DEC2 A4F63467 B4824CE0 E63ADAD3 D12A9C84 23437CE7
3163B310 227630A0 B3084773 E76D20A0 89EFDE03 783FC361 1C0B4F95 74577004 BE4AA329 D59A80AA
CDA0E335 EF37D8C5 C57B0E97 9C3E65A9 9124B209 C1796BE2 E597EEC1 FB534627 6B125767 0057BB2F
74974C59 4858E3E8 D927E037 EA0F522B A85517F5 74967031 F3C813B8 E071DAD2 60909173 7A383C02
AD45A093 DB0AC946 E7DD6528 8997FAB1 0F0E76D2 5EE75D47 78388568 EC23AAA9 EA7B482A 0E056708
58F41B48 B62D696B D23E72E5 2952B2A4 C11D49D4 D1AEF137 D31F3A7D 727B5A0D 62054C02 A8ECA1CC
B9922714 5E82D0E2 38D38927 EB348A6C 2805F19B 4151C12F D1F9FD9D 2FAA0092 5BF270D6 8CCBC59D
B2504D56 87019AB2 1B7FEE43 96B2E46B 80687E39 22D7E287 E8DD9071 341F7569 1C54BB3C 97EAB3F9
66335DCD 05BC1679 024B60BE F708DBE4 C624BB12 0E9BC09C 3D5EB6A8 ECEE0DED C5373FF4 F89D1425
4D0BBE0D 39D40954 85A90171 110E8660 365626A3 000C3F93 C93BD894 9B1BBB61 D05F7618 EABC81B3
57682EE9 4F60A488 3AE3F076 53264160 35AF2CF3 F27E5BCD F041ECC6 1A24404A A0870183 F2F9C891
2CB3B433 611F2389 264A500B D6F7C8EF 4C67B052 B8433C78 9BFCDAE2 ECC43EA8 BC6EC026 C453C011
0121C85C FAC9751B 51DC6A15 3C476E67 A0E074F0 3098DDC5 7A95BBDB 2F9E3D4F 3A560BC5 4911AE0B
DDE393F3 D7349EA8 EE064EB9 D3D918E6 37DD3C99 DACC5730 DE8637A5 DE960DA8 E733EA03 FD489F70
16496DC3 7179B88A 0F27EF9D 16FAEF6B B6D2A290 B3B63CC3 3183D83C 973D0BEE A83183BC D026C87D
77B7D076 4BFD0584 0EB02505 82D95523 3EF6AC92 1F2B90BF A596BCD0 0A09C227 03ACE97B DBAA544A
2E4403FE 5E349A54 417A6B71 E16C6080 2937917C 1D0B75C1 884AF5AF 74B7943D 5CA5A12D 5EF687CB
47E15C55 CD147D52 34863487 74B4E8F3 2F486BBA E370E418 EE2C1F86 C7863CD0 0C1138C4 9C2193FC
C7BD7043 C483F372 69520114 38841D3B A5CFBF28 B992273C DB611FC9 BBD1B946 B0594F90 AD48DF79
26DDF084 EE12B903 AF3A1EC2 18B5D412 7D7F7DBC 1DDABF0A 6ABF3DD4 81D90B07 AF574946 2FBF465C
20057F73 92C4F5EC CBD8A89D E7654214 FE9D0A72 3FE9F5A8 F0672F43 9EFF51BA 6F5AEB6A AD0C358B
4EAFA55D BF662991 65C4561C 88DF46C4 C95E8F8E 02709BA9 B09270DC 0E3F8FC5 DD78F973 D0B0861B
8DC2840A CEF1CB5A 5ECC06D5 29502642 A01B313D 06F796E0 B646111B E2601DBF BA55D173 0F877713
3B4AC9C6 AAC892FD EEF3DFC5 9278D214 9DF52F52 BC5376FB AFCC113B 3E3CB943 F3A1BADE 95D1A6E0
8063A180 AE996F3D 107A02AB 8444ACA1 E023A609 075C148F 2F49D740 004C7BAF C3C474C9 1DD0830B
```