Winternitz Abstracted Merkle Signatures (WAMS)

Author: Herman Schoenfeld
Version: 1.0
Date: 2020-07-20

Abstract

A quantum-resistant, many-time signature scheme combining Winternitz and Merkle signature schemes is proposed. This construction is compatible with the Abstract Merkle Signature scheme (AMS) [1] and thus is an AMS-algorithm called "WAMS".

1. Introduction

WAMS is a specialization of the AMS [1] scheme parameterized with the standard Winternitz one-time signature scheme (W-OTS). WAMS is the quantum-resistant cryptographic scheme used within the VelocityNET P2P platform.

This document focuses on the OTS-layer aspect of WAMS. Merkle-Signature Scheme aspects are performed in the AMS-layer of WAMS which is defined in the AMS document [1]. The reader should familiarize themselves with the AMS document as it provides the background and context for AMS-algorithms of which WAMS is one.

2. WAMS Scheme

The Winternitz Abstracted Merkle Signature Scheme (WAMS) is a general purpose, quantum-resistant signature scheme.

WAMS is an AMS algorithm that selects the standard Winternitz scheme (W-OTS) as the underlying the OTS scheme. As part of the parameter set inherited from AMS, WAMS includes the additional parameters `H` a cryptographic hash function and `w`, the Winternitz parameter.

The cryptographic hash function used is fundamental to the security of WAMS (an analysis of which is not provided in this document). So long as the user selects a standard CHF such as `SHA2-256` or `Blake2b` the security of WAMS follows that of other equivalent standard W-OTS constructions. For performance, the use of `Blake2b-128` can be used with an (acceptable) lowering of security for blockchain/DLT applications.

In this construction, the Winternitz parameter `w` refers to the number of bits being simultaneously signed as famously proposed by Merkle [2] (who was inspired by Winternitz). Varying the parameter `w` changes the size/speed trade-off without affecting security. For example, the higher the value the more expensive (and slower) the computations but the shorter the signature and private key size. The lower the value the faster the computation but larger the signature and key size. The range of values for `w` supported in WAMS is `1 <= w <= 16`.

Since the WAMS scheme inherits the AMS scheme, it is required to define the following:

- The OTS private key which is a standard W-OTS private key.
- The OTS public key is a standard W-OTS public key (hash).

- Definitions for `GenOTSSig` and `VerOTSSig` which generate and verify W-OTS signatures in accordance to the WAMS [1] specification.

Definitions for all of the above are provided below.

2.1 Notation & Definitions

1. Notations and definitions from AMS [1] are inherited by this document.
2. `ReadBits(arr, N, M)` is a function that skips `N` bits and then reads `M` bits from the byte array `arr` and re-interprets the bits as a big-endian unsigned 32-bit integer.
3. `WriteBits(x, arr, N, M)` is a function that converts unsigned 32-bit integer `x` to big-endian byte array of 4 bytes and writes the first `M` bits of the array into array `arr` start at bit offset `N`.
4. Bit-ordering in (2) and (3) is such that bit `i` of `arr` maps to byte `arr[i SHR 3]` and to in-byte bit-index `(i SHR 3) - (i SHL 3)`. Explained below:

```
Bit-ordering within `ReadBits` and `WriteBits` are such that the
   least-significant bit (LSB) is the left-most bit of that byte.

   For example, consider an array of two bytes C = [A,B]:

   Memory layout of C=[a,b] with their in-byte indexes marked.
   A:[7][6][5][4][3][2][1][0]  B:[7][6][5][4][3][2][1][0]
   C:[0][1][2][3][4][5][6][7]    [8][9]...

   The bit indexes of the 16-bit array C are such that:
     Bit 0 maps to A[7]
     Bit 1 maps to A[6]
     Bit 7 maps to A[0]
     Bit 8 maps to B[7]
     Bit 16 maps to B[0]
```

2.2 WAMS Parameters

Parameters	Description	Bits
`h`	Tree height (used in AMS layer)	8
`w`	Winternitz parameter, how many bits are simultaneously signed via the Winternitz process	8
`H`	Cryptographic hash function, and security parameter for the scheme (digest length)	8

Note that the Winternitz `w` and `H` are stored in the RESERVED part of the AMS private key. The cryptographic hash function is stored as a code, defined as follows:

2.2.1 Cryptographic Hash Function Code

Value	Cryptographic Hash Function
0	*user specified*
1	`SHA2-256`
2	`Blake2b-256`
3	`Blake2b-160`
4	`Blake2b-128`

The author reserves the right to update this list as new use-cases emerge.

2.3 WAMS Variables

During key generation, signing and verification the following variables are calculated based on the parameter set.

Variable	Formula	Description
`U`	`sizeof(H(x)) * 8` for any x	Security parameter for the scheme (and number of bits in a hash `H`)
`DigitBase`	`2^w`	The number of values a signed "digit" can take
`SigDigits`	`ceil(256 / w)`	Number of digits in the message-digest being signed
`CheckDigits`	`Log((2^w - 1) * (256/w))_{2^w}`	Number of digits in checksum being signed
`OTS_KeyDigits`	`SigDigits + CheckDigits`	Number of "digit keys" in a W-OTS private key (used by AMS-layer)
`OTS_SigLen`	`OTS_KeyDigits`	Number of "digit signatures" in a W-OTS sig (used by AMS-layer)

2.4 W-OTS Theory Basics

The W-OTS scheme follows the Lamport [3] signature approach but allows a signer to sign `w` bits of a message-digest simultaneously rather than 1. This collection of bits is a treated as a "digit" of base `2^w`.

For example, in the case of `w=8` the digits simply become bytes since each digit can take any value within `0..255`. The fundamental cryptographic mechanism in W-OTS is the ability to sign individual digits using a unique "digit private key".

For example, to sign the byte `b` (for `w=8`), a signer first derives a "private digit key" as `K = H(secret)` and a "public digit key" `P = H^255(K)`. Notice that all the values of `b` map to a unique hash in that chain of hashes. The signer advertises the "public digit key" prior to signing any digit. When signing a digit `b`, the signer provides the verifier the value `S=H^(255-b)(K)` referred to as the "signature of `b`". The verifier need only perform `b` more iterations on the signature `s` to arrive at the public key `P`, since `H^b(S) = H^b(H^(255-b)(K)) = H^255(K) = P`.

At this point, the verifier has cryptographically determined the signer had knowledge of `K` since the signature `S` was the `b'th` pre-image of `P`. This process of signing digits is repeated for each digit in the message and each digit signature concatenated to form the signature. The message being signed is always a digest of an actual logical message, and thus referred to as the "message-digest".

In W-OTS, the individual "digit keys" and "digit signatures" are concatenated to comprise the "key" and "signatures" respectively. This results in order of magnitude larger keys and signature objects than typical elliptic-curve or discrete logarithm cryptography schemes. This is a significant down-side of OTS schemes used in post-quantum cryptography (PQC). The burden of large keys can be optimized by using the public key hash as WAMS prescribes. The burden of large signatures can be halved by choosing shorter hash functions without impacting security, as prescribed by the W-OTS#[4] variant. .

NOTE In order to prevent signature forgeries arising from digit signature re-use for prior messages, a checksum is calculated and appended to the message-digest and co-signed. The checksum is calculated in such a way that any increment to a message digit necessarily decreases a checksum digit. Thus it is impossible to forge a signature since it requires the pre-image of at least one checksum digit signature.

The reader can further their understanding of the theory and basics of W-OTS by reviewing the literature and through this succinct diagram[5].

2.4.1 W-OTS Private Key

A W-OTS private key `P'` is a one-time key used to generate W-OTS signatures and defined as follows:

```
1: P' = byte-array[OTS_KeyDigits, U/8]
2: for n in {0, OTS_KeyDigits - 1}
3:    P'[n] = cryptographically random U bits
```

The W-OTS private key is an array of `OTS_KeyDigits` "digit keys" each of `U/8` bytes in length. The total size of the W-OTS private key is thus `(OTS_KeyDigits) * (U/8)` bytes.

Whilst the W-OTS scheme requires that private keys be cryptographically random, they can be deterministically derived from a secret seed. In WAMS the AMS Private Key is used (see below).

2.4.2 W-OTS Public Key Hash

A W-OTS public key hash `K'` is a one-time key used to verify W-OTS signatures signed by a W-OTS private key `P'` and defined as follows:

```
1: k = byte-array[OTS_KeyDigits, U/8]
2: for n in {0, OTS_KeyDigits - 1}
3:    k[n] = H^(DigitBase - 1)( P'[n] )
4: K' = H( k[0] || k[1] || ... || k[OTS_KeyDigits - 1] )
```

The length of a W-OTS public key hash is `U/8` bytes.

NOTE In WAMS, the W-OTS public key hash is used rather than the W-OTS public key since signature verification always rebuilds the public key from the signature. Since the verifier derives the public key it can derive the public key hash with one additional step. By using the hash rather than the key in the AMS signature, a ~50% space saving is made to the AMS signature length.

NOTE 2 Since the OTS layer passes the public key hash to the AMS layer, the AMS layer does not need hash the public keys when building the hash-tree of OTS keys, it simply re-uses the OTS public key value which is itself a hash digest (saving `2^h` hash computations when computing a batch).

2.5 WAMS Key Generation

Given a AMS Private Key `P` and batch number `B`, the `i'th` W-OTS key-pair (`P'`, `K'`) are derived as follows:

```
 1: algorithm GenOTSKeys
 2:   Input:
 3:         P: AMS Private Key
 4:         B: batch number (UInt64)
 5:         i: index (UInt16)
 6:   Output:
 7:         P': the W-OTS private key that derives K'
 8:         K': the i'th W-OTS public key hash in the batch
 9:   Pseudo-Code:
10:     P' = byte-array[OTS_KeyDigits, U/8]
11:     k = byte-array[OTS_KeyDigits, U/8]
12:     let seed = ToBytes(i) || ToBytes(B) || P
13:     for n in {0, OTS_KeyDigits - 1}
14:        P'[n] = H^2( n || seed )
15:        k[n] = H^(DigitBase - 1) ( P'[n] )
16:     K' = H( k[0] || k[1] || ... || k[OTS_KeyDigits - 1] )
17: end algorithm
```

2.6 W-OTS Signature Generation

A W-OTS signature is an 2D array of bytes of dimensions `[OTS_KeyDigits, U/8]` and generated as follows:

```
 1: algorithm GenOTSSig
 2:   Input:
 3:         m: a message-digest (U/8 bytes)
 4:         P': a W-OTS private key
 5:   Output:
 6:         S': a W-OTS signature
 7:   Pseudo-Code:
 8:     S' = byte-array[OTS_KeyDigits, U/8]
 9:     // sign message part
10:     let c = 0                      ; checksum value
11:     for n in {0, SigDigits - 1}
12:        let v = 2^w - ReadBits(m, w*n, w) - 1
13:        c = c + v;
14:        S'[n] = H^v( P'[n] )
15:
16:     // sign checksum part
17:     let c_bytes = byte-array[4]
19:     WriteBits(c, c_bytes, 0, 32)
20:     for n in {0, CheckDigits - 1}
21:        let v = 2^w - ReadBits(c_bytes, w*n, w) - 1
22:        S'[SigDigits + n] = H^v( P'[SigDigits + n] )
24: end algorithm
```

2.7 W-OTS Signature Verification

Here a W-OTS signature is verified to a W-OTS public key hash by rebuilding the W-OTS public key from the signature, hashing it and comparing with public key hash provided by the AMS layer.

```
 1: algorithm VerOTSSig
 2:   Input:
 3:        S': a W-OTS signature (byte[ OTS_KeyDigits, U/8 ])
 4:        m: a message-digest (byte[U/8])
 5:        K': W-OTS public key/hash (byte[U/8])
 6:   Output: Boolean
 7:   Pseudo-Code:
 8:        k = byte[ OTS_KeyDigits, U/8 ]    ; the W-OTS public key
 9:        ; verify message part
10:        let c = 0                         ; checksum value
11:        for n in {0, SigLen - 1}
12:           let d = ReadBits(m, w * n, w)  ; note: d + v = 2^w - 1
13:           c = 2^w + d - 1
14:           k[n] = H^d( S'[n] )            ; note: k[n] = H^d(H^c(P'[n]))
15:
16:        ; verify checksum part
17:        let c_bytes = byte-array[4]
18:        WriteBits(c, c_bytes, 0, 32)
19:        for n in {0, CheckDigits - 1}
20:           let d = ReadBits(c_bytes, w * n, w)
21:           k[SigDigits + n] = H^d( S'[SigDigits + n] )
22:
23:        ; compare pub key hash
24:        let PKH = H( k[0] || k[1] || ... || k[OTS_KeyDigits - 1] )
25:        return (K' = PKH)                ; check sig rebuilt the public key
hash
26: end algorithm
```

3. WAMS#

WAMS# is a variant of WAMS which selects W-OTS#[4] rather than W-OTS as the OTS. W-OTS# is virtually identical to W-OTS except the message-digest is salted to harden the signature security to a sufficient level that thwarts birthday-class attacks. This allows the selection of shorter hash functions which produce shorter and faster signatures for same security as W-OTS.

The WAMS# implementation is virtually identical to WAMS except for the following changes:

1. A cryptographically random salt `R` of `U`-bits is generated during signing.
2. For any message `m`, the signer signs the "sig-mac" `SMAC(m, R)` rather than the message-digest `H(m)` which is defined as `SMAC(m, R) = H(R || H (R || H(m)))`.
3. `R` is appended to the signature.
4. During verification, the verifier similarly verifies `SMAC(m, R)` rather than the ordinary message-digest.

The reader is referred to the reference implementation of WAMS# which succinctly overloads WAMS with these minor changes.

4. Object Lengths

Throughput is normalized to the throughput of configuration `W-OTS w=256, h=0` which is the standard-bearer for PQC. Thus they give a relative measure of performance and interpreted as "how many times faster than standard W-OTS". On the test machine, this base configuration signed XXX messages per second and verified YYY per second.

OTS	CHF bits	Winternitz `w`	Height `h`	Public Key Length (b)	Signature Length (b)	Sign Throughput	Verify Throughput
W-OTS	128	2	1	32	2163		
W-OTS#	128	2	1	32	2211		
W-OTS	128	2	8	32	2163		
W-OTS#	128	2	8	32	2211		
W-OTS	128	2	16	32	2163		
W-OTS#	128	2	16	32	2211		
W-OTS	128	4	1	32	1107		
W-OTS#	128	4	1	32	1155		
W-OTS	128	4	8	32	1107		
W-OTS#	128	4	8	32	1155		
W-OTS	128	4	16	32	1107		
W-OTS#	128	4	16	32	1155		
W-OTS	128	8	1	32	579		
W-OTS#	128	8	1	32	627		
W-OTS	128	8	8	32	579		
W-OTS#	128	8	8	32	627		
W-OTS	128	8	16	32	579		
W-OTS#	128	8	16	32	627		
W-OTS	160	2	1	36	2703		
W-OTS#	160	2	1	36	2763		

OTS	CHF bits	Winternitz `w`	Height `h`	Public Key Length (b)	Signature Length (b)	Sign Throughput	Verify Throughput
W-OTS	160	2	8	36	2703		
W-OTS#	160	2	8	36	2763		
W-OTS	160	2	16	36	2703		
W-OTS#	160	2	16	36	2763		
W-OTS	160	4	1	36	1383		
W-OTS#	160	4	1	36	1443		
W-OTS	160	4	8	36	1383		
W-OTS#	160	4	8	36	1443		
W-OTS	160	4	16	36	1383		
W-OTS#	160	4	16	36	1443		
W-OTS	160	8	1	36	723		
W-OTS#	160	8	1	36	783		
W-OTS	160	8	8	36	723		
W-OTS#	160	8	8	36	783		
W-OTS	160	8	16	36	723		
W-OTS#	160	8	16	36	783		
W-OTS	256	2	1	48	4323		
W-OTS#	256	2	1	48	4419		
W-OTS	256	2	8	48	4323		
W-OTS#	256	2	8	48	4419		

OTS	CHF bits	Winternitz `w`	Height `h`	Public Key Length (b)	Signature Length (b)	Sign Throughput	Verify Throughput
W-OTS	256	2	16	48	4323		
W-OTS#	256	2	16	48	4419		
W-OTS	256	4	1	48	2211		
W-OTS#	256	4	1	48	2307		
W-OTS	256	4	8	48	2211		
W-OTS#	256	4	8	48	2307		
W-OTS	256	4	16	48	2211		
W-OTS#	256	4	16	48	2307		
W-OTS	256	8	1	48	1155		
W-OTS#	256	8	1	48	1251		
W-OTS	256	8	8	48	1155		
W-OTS#	256	8	8	48	1251		
W-OTS	256	8	16	48	1155		
W-OTS#	256	8	16	48	1251		

4. Reference Implementation

See https://github.com/Sphere10/Hydrogen/tree/master/src/Sphere10.Framework/Crypto/PQC.

References

1. Herman Schoenfeld. "Abstracted Merkle Signatures (AMS)", 2020. ↩ ↩ ↩ ↩ ↩

2. Ralph Merkle. "Secrecy, authentication and public key systems / A certified digital signature". Ph.D. dissertation, Dept. of Electrical Engineering, Stanford University, 1979. URL: http://www.merkle.com/papers/Certified1979.pdf. ↩

3. L. Lamport. "Constructing digital signatures from a one-way function". Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, Oct. 1979. ↩

4. Herman Schoenfeld. "W-OTS# - Shorter and Faster Winternitz Signatures". URL: https://vixra.org/abs/2007.0194. Accessed on: 2020-07-20. ↩ ↩

5. Crypto4A. "Hash Chains and the Winternitz One-Time Signature Scheme". URL: https://crypto4a.com/sectorization-defunct/W-OTS/. Accessed on: 2020-07-20. ↩