

Identity Aware Concise Hash Based Signatures

DRAFT Idea in progress - Do not distribute

Laurent Henocque, Gabriel Risterucci
Keeex - <https://keeex.me>
laurent@keeex.net, gabriel@keeex.net

May 7, 2026

Abstract

This document describes a possible (yet unproven) method for making a quantum-resistant Winternitz-type one time signature (WOTS or WOTS+) more compact. It builds upon the intuition that the entropy of such signatures should not exceed the entropy of signed messages (typically 256 bits) by orders of magnitude. Two distinct schemes are proposed, the first one being very close to the intuitions in WOTS, named CWOTS for Concise WOTS, and the second called ICWOTS for Identity aware CWOTS. Both schemes are usable in stateful signature schemes.

ICWOTS introduces Identity by using open hash chains that enable unlimited signatures by identified signatories.

Support for quantum resistant ZKStark based stateless signature schemes are not considered since as per our knowledge quantum resistant zero knowledge proofs are very lengthy and “compact” signatures are sought.

Disclaimer : No cryptanalytic analysis of the intuitions presented in this text has been conducted.

1 Introduction

This document investigates a way to make a quantum-resistant Winternitz-type OTS (One-Time Signature) (as WOTS and WOTS+¹) more compact. A second motivation is to explore the possibility of using WOTS+ in a quasi stateless signing scheme allowing for producing an unlimited number of quantum resistant hash based signatures. Regarding size in particular, in the approach proposed herein that considers messages and keys with length of 32 bytes, the size of a signature is reduced to 64 bytes, regardless of the chosen winternitz parameter w . Among motivations for this study is the potential attack over ECDSA signatures by quantum cryptography in a foreseeable future.

For use on the Bitcoin blockchain, one time signatures may be sufficient because using payment addresses only once is already recommended, and forms the primary purpose of BIP32, that enables addresses to never be reused through the use of change addresses. The purpose of this paper is not to propose a possible improvement to Bitcoin : this example is meant to suggest the existence of at least a use case for short length one time signatures.

¹<https://di-mgt.com.au/pqc-03-winternitz.html>

Note however that in the context of blockchain in general, quantum-resistant signatures provide the only way to prevent “front-running” quantum attacks that are possible when an ECDSA signature and thus its matching public key have been revealed. Quantum attacks aiming to discover the private key and front run a transaction could occur while it is public in the mempool.

In the context of long standing digitally signed documents or transactions, signers own or represent identities that remain consistent over time and are willing to generate numerous signatures using the same identity. There an attacker has an unlimited amount of time to impersonate an identity, since many signatures are revealed through static, long lasting documents.

Signatures are called “concise” here because unlike with standard Winternitz schemes (the reader is assumed to be familiar with this), the function chains considered do not each produce a 32 byte output but instead a single byte. This is achieved by using an additional 32 byte seed.

1.1 Settings

The sequel will focus on 256-bit integers, that may be divided into 32 elementary bytes (whose values range from 0 to 255). This corresponds to using a Winternitz parameter of 256 (or 8 if $\log_2(256)$ is preferred).

As with WOTS, it is also possible to split 256 bits into 64 4-bit integers ranging from 0 to 15 or 128 2-bit integers (0 to 3). Even thinkable are the options of splitting 256 bits into 16 16-bit words and even 256 1-bit words. At this stage, no performance-versus-security metrics are known. Adapting the scheme to whichever parameter is straightforward. The trade-off concerns the number of hash iterations (up to 256) and thus implicitly security. As will be seen, this particular choice will not impact the length of signatures.

In the sequel *Hash* occurs as a generic name that may denote a suitable NIST approved hash function (as e.g. SHA256 or SHA3,), a multi-hash function like KeeX Multi Hash² and/or a use of HMAC using one of what precedes with an extra parameter.

1.2 Plan

The plan is as follows : a first section shall introduce a first version of one time signatures that highlights the role of a 32 byte seed parameter in the definition of 64 byte one time signatures, called Concise WOTS or CWOTS. Such signatures can be bound to an identity using a stateful approach via Merkel trees as in WOTS+ (MSS).

A second section will introduce Open Hash Chains : an identity tracking scheme offering potentially unlimited numbers of signatures from the same identity.

A third section will introduce the ICWOTS scheme where the binding between an identity and signatures is made explicit and puts to work the results from section 2.

²<https://docs.keeex.me/en/specs/multihash>

2 Concise One Time Signatures CWOTS

2.1 Private key

A private key is a randomly generated number or the hash of a secret, represented as a 32-byte sequence.

$$P = p_0 \dots p_{31}$$

2.2 Public key

2.2.1 Seed

We first randomly or pseudo randomly choose a seed S .

$$S = s_0 \dots s_{31}$$

2.2.2 Computing public key bytes

We assume the choice of a Hash function called *Hash*. We now compute for every byte number i in $(0, \dots, 31)$ the following iteration:

$$H_i^0 = \text{Hash}(s_0 \dots s_{i-1} p_i s_{i+1} \dots s_{31}) = h_0 \dots h_i^0 \dots h_{31}$$

where h_i^0 denotes the i^{th} byte in H_i^0 . Then

$$H_i^k = \text{Hash}(s_0 \dots s_{i-1} h_i^{k-1} s_{i+1} \dots s_{31}) = h_0 \dots h_i^k \dots h_{31}$$

and finally

$$H_i^{255} = \text{Hash}(s_0 \dots s_{i-1} h_i^{254} s_{i+1} \dots s_{31}) = h_0 \dots h_i^{255} \dots h_{31}$$

2.2.3 Merging the public key

The public key now defines as the merging of all h_i^{255} in one 32 byte integer :

$$\text{Pubk}_S = h_0^{255} \dots h_{31}^{255}$$

2.2.4 Assembling the Public Signing Capability Record

In order to prevent attempts to brute force a seed after a key was shared, the seed hash may be included in the information provided to a future verifier. We call this information the Public Signing Capability Record (PSCR), that consists in the pair $\text{Pubk}_S, \text{Hash}(S)$.

$$\text{PSCR}_S = \langle \text{Pubk}_S, \text{Hash}(S) \rangle$$

2.3 Message signing

Given a message M , the $PSCR$ and $HMAC(Hash(PSCR), Hash(M))$ being represented as $m_0...m_{31}$ the cryptographic part of the signature is defined as the 32 byte integer :

$$Sig_S(M) = h_0^{m_0}...h_{31}^{m_{31}}$$

Since the seed S is required for verification and was not revealed as part of the $PSCR$, it is published as part of the full signature.

$$Signature(M, S) = \langle Sig_S(M), S \rangle$$

The seed in the signature can be verified against the hash present in the $PSCR$. The signature for a given message hence counts 64 bytes under our hypotheses. The motivation for using $HMAC(Hash(PSCR), Hash(M))$ as an input to signing instead of $Hash(M)$ alone is explained below.

2.4 Message signature verification

Given a message M , a signature Sig_M , a seed S and a public key $Pubk_S$ the verification attempts to reconstruct the public key. It proceeds by applying the same public key computation procedure to every i^{th} byte in Sig_M , starting at position m_i and using S , and then checking that the value of each obtained h_i^{255} matches the i^{th} byte in $Pubk_S$.

2.5 Checksum

As with WOTS the above process is subject to brute force attacks attempting to produce a message where all bytes are bigger than the original m_i so that a valid signature can be inferred for a known public key. This risk is mitigated by adding checksum bytes as proposed in WOTS+ and SLH-DSA³.

Given a signed message the checksum is computed as the sum of all $255 - m_i$ for $0 \leq i < 32$. The idea is that should a message have all its m_i bytes bigger than those in a previously valid signed message, the checksum would have at least one byte lower, which prevents any replay attack on the signature (since revealing any m_i also reveals all subsequent chain bytes). This checksum is less than $255 * 32 = 8160$ hence can be encoded as a two byte integer.

Existing frameworks suggest adding extra words to the original public key, and to signatures as well. Despite the loss of two bytes of entropy, we choose here to keep the length of 256 bytes to both the public key and signature, by sacrificing the last two bytes for the checksum. We therefore compute the checksum over the first 30 bytes and call the corresponding two checksum bytes sum_1 and sum_2 .

Given a 32 byte message M , and $Hash(message) = m_0...m_{31}$ the signature is now defined as :

$$Sig_S(M) = h_0^{m_0}...h_{29}^{m_{29}}, h_{30}^{sum_1}, h_{31}^{sum_2}$$

and the verification proceeds by verifying accordingly : computing the checksum from the 30 first message bytes, then generate the final two, then reconstruct the public key.

³<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf>

2.6 Signing the PSCR

Beyond the need for a checksum, another problem may arise. Once knowledge of a seed was published (by a valid signature), an attacker may pretend having that knowledge. The seed shall be used in the sequel for binding identities to keys and signatures. Pretending having knowledge of a seed will therefore amount to pretending having an identity, which requires to circumvent this issue.

Assuming an attacker wishes to sign a message using an already revealed seed, they can generate very easily a public key that matches the message by picking random $h_i^{m_i}$ values and recomputing the public key like the verification does. The corresponding PSCR can be shared as genuine, and the signature as well.

The countermeasure is to let the signature depend upon the PSCR. To enforce this permanently (even though all or part of the PSCR may be mentioned manually by a user in a signed message) the bytes acting as selectors in the public key chains are those of $HMAC(Hash(PSCR), Hash(M))$.

2.7 Discussion

Because disclosing a signature reveals every $h_i^{m_i}$ and consequently all subsequent h_i^j for $m_i < j \leq 255$ a public key should never be reused, hence is “one time”. Such one time signatures can be involved in XMSS or hypertree schemes⁴ in order to expose identities matching a possibly large number of public keys. Sections 2 and 3 below will explore a novel approach to resusing the private key across multiple signatures as well as enforcing verifiable identity.

2.7.1 Variants

The public key computation process may hash S and/or rotate S bytes at every iteration. The rotation value may be 1 bit at a time (which rotates the 256 bits at chain end) or any other deterministic value (h_i^0).

It can be explored whether the use of the $\oplus$ operator (exclusive or) could replace or extend hash computations for further efficiency / security tradeoffs.

Again no cryptanalysis was performed on the whole scheme or such variants.

3 Open Identity Tracing OIT

Identity enforcement in signature schemes requires guaranteeing that a sequence of signatures originates from the same signatory. This can be achieved by a proof of knowledge of a secret, for instance by revealing a hash preimage. Unless in large volume industrial situations, the number of signatures a “standard” signer may wish to create remains limited.

A simple way to bind signatures together is to precompute a fixed length hash chain (where each element is a preimage of the next), starting from a chain secret C , by iterating calls to a hash function $Hash(Hash(...(C)))$ and then reveal its final element as a definition of identity, to be published as part of a PSCR. Every signature will reveal an intermediate element in the chain, thereby proving knowledge of a preimage of the identity. Revealing elements

⁴<https://di-mgt.com.au/ppc-10-ht-sig.html>

is made in decreasing order, until the list would be exhausted to the point of revealing the 'head' preimage C .

Another choice is to compute a Merkle tree from a number of leaves and reveal it's "root" defining an identity as part of a PSCR. Every signature will reveal a leaf of the merkle tree together with the matching merkle branch, again proving knowledge of a preimage of the Merkle "root".

The two approaches share a common difficulty : the number of allowed signatures is pre-determined, which is unsuitable when anticipating usage is impossible.

For simplicity, also because Merkle trees are somehow "overkill" for small numbers, and because they add the cost of a merkle branch to every signature we focus on hash chains in the sequel. (Both options are can be used anyway, even in combination.)

3.1 Open Hash Chains

To overcome the fixed signature count limitation, we need to enable an identity to iteratively use multiple hash chains over time, under an "on demand" strategy. This subsection introduces "Open" hash chains for this purpose.

For every $k > 0$ we note $Hash^k(x)$ the repeated application of the function $Hash$ to its argument ($Hash(Hash(...(x)))$ k times). This creates a Hash chain.

All hash chains used by an identity in what follows are started from chain starts denoted as $C_0, C_1, \dots$ and have lengths $l_0, l_1, \dots$. For simplified private material handling by the signer the sequence of C_i may be privately deterministic, so that knowing C_0 suffices to regenerate the entire signing private data.

3.1.1 Defining Identity

Identity will be defined as the final element of the first chain started at C_0 . Given l_0 as the chain length, we define H_0^0 as $Hash(C_0)$ and for every $0 < k \leq l_0$, the k^{th} element in the chain is :

$$H_0^k = Hash^k(H_0^0)$$

and we then define the identity bound to C_0 as the final element in the chain :

$$Id_0 = Hash^{l_0}(H_0^0)$$

3.1.2 Assessing Identity

Then, every request for proving identity, for instance as part of a signature, involves revealing a new element in the chain, that when iteratively hashed reconstructs the identity (and all the elements in between). Items must hence be revealed in reverse order, the last to be usable being the root of the hash chain (hence H_0^0 for chain 0). In revealing an H_0^k an actor does not reveal H_0^j for any $0 \leq j < k$, but proves knowledge of a preimage of Id_0 .

This approach is viable when a limited number of signatures may be expected from an identity.

3.1.3 Using Chain nodes as seeds

In relation to the previously presented signature scheme, any H_0^k can be used to derive the seed S , and C_0 may also be privately used to derive the private key. Under those settings C_0 therefore acts as a master secret. Note however that in extreme cases proving knowledge of the preimage of H_0^0 remains possible, either explicitly or via zero knowledge proofs.

3.1.4 Last signature is for extension

To enable extending the range of signatures made possible by a known identity we state that the revelation of H_0^0 should never be made as part of a usual signature, but only for signing a statement declaring a continuation chain ($C_1, \dots$).

3.2 Extending by adding a new chain

We can design an unlimited signature scheme from this by using the first entry in the chain C_0 . In case when more than l_0 signatures are needed for an identity, H_0^0 can be used to as part of a signature of a continuation chain, derived from a new hash chain with a new secret C_1 .

Using H_0^0 as a seed yields the PSCR used to sign a simple message that simply mentions the current chain index and the new hash chain termination. :

$$< O, Id_1 >$$

The process can be repeated without limit. Using this approach, every hash chain added to C_0 will add a payload to PCSRs but not to signatures themselves.

When the current hash chain is exhausted, the signers and signing systems face the choice of arbitrating between keeping the same initial identity ($Id_0 = Hash^{l_0}(H_0^0)$) using a new extension chain or declaring a new identity.

It should be noted that to attach digital identities to real world actors usually requires to securely disclose public keys (here PCSRs) behind a web domain, which supports adding new identities.

4 Identity Aware Concise One Time Signatures ICWOTS

We now present a variation of the CWOTS scheme, where an identity constructed from C_0 can possibly be bound to an unlimited number of signatures using the previous procedure. The main objective is to reuse the private key over many signatures by letting the seed vary. The second objective is to enable unlimited signatures by a given identity.

Under the OIT terms the same pair $< chain\ id, chain\ entry\ index >$ should never be reused, and the chain entry index must be decreasing with every signature, from the chain length minus one to 1. Both indexes 0 and 1 have special usage. The final index 1 refers to the identities, and the index 0 is used to sign extension chains.

4.1 Private key

A private key is a 32-byte integer $Priv$. Since the private key shall be reused over all the signatures this value forms the defining part of the identity. $Priv$ can possibly be derived from C_0 or the opposite, or the two may remain unrelated. Derivation options are secret and must not interfere with other results.

$$Priv = p_0 \dots p_{31}$$

4.2 Private key derivation

As a primary marker of identity (and hypothetical exceptional proofs of knowledge of the private key), a derivation of $Priv : Hash(Priv)$ shall be added to the PSCR. Technical usage of $HPriv$ in the public key derivation may be considered but is not analyzed.

$$HPriv = Hash(Priv)$$

4.3 Identity

Id_0 must be added to the PSCR

4.3.1 Seed

We choose the seed S as follows. For the current pair of a chain id c and chain entry index k we pick

$$S = H_c^k$$

By deterministically choosing S we enforce the fact that every node in the hash chain can be used only once and also prevent from brute force attacks that could proceed by enumerating seeds.

4.3.2 Computing the Public key bytes

This is performed as in CWOTS, section 1 above.

4.3.3 Merging the Public key bytes

As in CWOTS the Public key now results from merging of all h_i^{255} in one 32 byte integer :

$$Pubk = h_0^{255} \dots h_{31}^{255}$$

4.3.4 PSCR

Publishing a signing capability record for an element in chain 0 now requires sharing the following tuple

$$PSCR(0, k) = \langle Pubk, Hash(H_0^k), HPriv, Id_0, 0, k, l_0 \rangle$$

(Note that $Hash(Seed = H_0^k) == H_0^{k+1}$)

$Pubk$ is the signing public key according to WOTS, $Hash(Seed)$ prevents enumerating seeds and in the current context identifies the previous signing level, $HPriv$ is the most stable identity marker, as the Hash of the private key, 0 identifies the chain 0 and k the chain node chosen. l_0 and Id_0 are used to prove that signatures reveal knowledge of a preimage of Id_0 .

Publishing a signing capability using chain $c \neq 0$ requires to additionally share the message and matching signature that declares the new chain, as described in section 2 OpenHash Chains above.

$$PSCR(c, k) = \langle Pubk, Hash(H_c^k), HPriv, Id_0, c, Id_c, k, l_c, extension\ sigs \rangle$$

4.4 Message signing

Given a message M , the declared $PSCR$ and $HMAC(Hash(PSCR), M) = m_0 \dots m_{31}$ the cryptographic part of the signature is defined as in CWOTS :

$$Sig_M = h_0^{m_0} \dots h_{29}^{m_{29}}, h_{30}^{sum_1}, h_{31}^{sum_2}$$

The published signature must disclose the pair $Seed, Sig_M$. This is necessary since the published PSCR only mentioned a hash of the seed :

$$Signature(M) = \langle Sig_M, H_c^k \rangle$$

The signature integrity may be verified against the public key by checking that :

- a/ the Seed H_c^k in the signature matches the Hash present in the PSCR
- b/ the Seed H_c^k can be hashed repeatedly upto Id_c (using k and length l_c).
- c/ extension sigs and details validate

4.5 Message signature verification

Given M, Sig_M , the seed S and the identity the verification proceeds by first checking that S is a valid antecedent to $Identity$ plus the other mentioned verifications, then by applying the public key computation procedure to every i^{th} byte in Sig_M , starting at position m_i and using S and $PSCR$, and then checking that the value of each obtained h_i^{255} matches the i^{th} byte in $Pubk$.

4.6 Discussion

Signatures have a limited size of two 32 byte words. PSCRs amount to a limited number of 32 byte words that increase with every newly added open chain branch. Optimal implementations may be searched in that area if the scheme passes cryptanalysis.

Because disclosing a signature reveals every $h_i^{m_i}$ and consequently all subsequent h_i^j for $m_i < j \leq 255$ a public key should never be reused, hence is “one time”.

Unlike fully stateless schemes, ICWOTS does not support that several unrelated entities use the same private key and identity. This is so because at every signature, the signer must decrease the value k or create a new Open Chain branch. Beyond this limitation, it was seen that a signer must only keep

a permanent record of the private key $Priv$, the open hash chain “root” C_0 and the current values k and c .

The described schemes only rely upon hash functions and are thus natively quantum resistant.

5 Conclusion

This document describes a possible implementation of hash based one time signatures in the WOTS spirit having a length of 64 bytes whichever granularity of sampling is chosen.

A first proposal called CWOTS is close to the choices made in WOTS, except that the size of information retained per byte in the private key is also one byte. This is made possible by using a secondary seed that provides entropy around the byte. This scheme supports the same extensions as WOTS+, including adding a checksum and integrating signatures within a stateful scheme. CWOTS also enables reusing the same private key across many signature by letting the seed vary.

A notion of Open Hash Chains is presented that supports a simple definition of Identity and unlimited signing capability by a given Identity.

The second proposal ICWOTS natively supports identity aware quantum resistant signatures. This proposal uses two identity markers, first by a published hash of the private key, then by using the final element in an open hash chain. This identity can be reused across an unlimited number of signatures by using seeds produced by the open hash chain.

As presented, the ICWOTS scheme aims to prevent signature reuse as well as identity theft.

Disclaimer : no cryptanalysis of the proposal was performed to date.