



# Towards Post-Quantum Distributed Validators on Ethereum

**Abstract:** With recent advances in quantum computing, post-quantum cryptographic algorithms are being actively deployed in real-world applications. For Ethereum, a transition to post-quantum cryptography would require replacing many primitives, including the BLS12-381 signature schemes used by validators on the beacon chain. Because BLS leverages its bilinear pairing property to aggregate multiple validator signatures, enabling both performance improvements and space savings, its replacement presents a particular challenge. Furthermore, the bilinearity of the pairing function also enables straightforward threshold signatures, which are fundamental to distributed validator solutions. The objective of this report is to review the current state of post-quantum signature algorithms and identify those most suitable for the Ethereum beacon chain, particularly focusing on their support for signature aggregation and threshold functionality.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Aggregate hash-based signatures using SNARKs</b>	<b>1</b>
2.1	Hash-based signatures . . . . .	1
2.2	SNARK-based aggregation . . . . .	2
<b>3</b>	<b>Towards threshold XMSS</b>	<b>2</b>
3.1	Distributed hash-based signatures with Boolean shares. . . . .	3
3.2	Leveraging secret-sharing . . . . .	3
<b>4</b>	<b>Hash functions over MPC</b>	<b>3</b>
4.1	Beaver triples . . . . .	5
4.2	MPC-friendly hash functions . . . . .	5

February 4, 2025

## 1 Introduction

If a cryptographically relevant quantum computer is built, Shor’s algorithm will pose serious threats to traditional public key cryptosystems based on large number factorization and (elliptic curve) discrete logarithm problems, such as RSA and ECDSA. As a response, cryptographers are developing new algorithms that offer security even against an attacker equipped with quantum computers, denoted as post-quantum cryptography (PQC). There are several standardization processes ongoing, notably by the one NIST which has already published a set of standards: ML-KEM [oST24b] as key encapsulation mechanism along with ML-DSA [oST24a], SLH-DSA [oST24c] and FN-DSA, to be published soon, as signature schemes. Because ML-DSA and FN-DSA are both lattice-based, NIST has solicited the submission of additional signatures schemes to expand its PQC signature portfolio<sup>1</sup>. Unfortunately, the current post-quantum signature schemes selected by NIST for standardization do not inherently support advanced functionalities such as signature aggregation and/or threshold signing. Signature aggregation is commonly used in blockchain systems as this powerful feature allows to compress many signatures into a short aggregate, shrinking the storage space and speeding-up the verification time. Ethereum leverages takes advantage of aggregate signatures in its consensus layer thanks to the BLS signature scheme [BLS01]. On top of intrinsically supporting signature/public key aggregation, BLS is straightforward to be turned into a threshold signature scheme when combined with Shamir Secret Sharing. An important observation in the case of BLS is that aggregated and/or threshold BLS signatures are indistinguishable from raw ones, all being points on the same elliptic curve. This allows to build efficient distributed validator solutions, such as the **charon** middleware<sup>2</sup> which operates in a totally transparent manner from a consensus client point of view. However, because BLS is based on elliptic curve pairing, it would not provide enough security against quantum adversaries. To address this concern, the Ethereum foundation recently introduced a family of hash-based signature schemes as post-quantum alternatives to BLS [DKKW25]. The main idea behind their design is to aggregate hash-based signatures using post-quantum succinct non-interactively argument of knowledge (pqSNARK) systems. While this seems to be a promising alternative, it might have considerable impacts on distributed validators solutions which currently take advantage of the homomorphic properties from BLS. The goal of this document is to identify the challenges that could arise from such a transition and discuss the potential solutions to address them.

## 2 Aggregate hash-based signatures using SNARKs

### 2.1 Hash-based signatures

As their name suggests, hash-based signature schemes rely on hash functions as their core primitive. In contrast to public key cryptosystems, there is no strong evidence that symmetric cryptography, including hash functions, would be significantly impacted by quantum computers. Although recommendations on symmetric cryptography may vary between cybersecurity agencies<sup>3</sup>, hash-based signatures are seen as a conservative choice for post-quantum security given their well-understood security. The classical approach to build hash-based signatures is to combine many one-time signature (OTS) key pairs into a Merkle tree [Mer79] whose root serves as the many-time public key. To provide a concrete example, we hereafter introduce the Winternitz OTS (WOTS) scheme.

<sup>1</sup><https://csrc.nist.gov/projects/pqc-dig-sig>

<sup>2</sup><https://github.com/ObolNetwork/charon>

<sup>3</sup>ANSSI recommends at least the same security as AES-256 and SHA2-384 for block ciphers and hash functions, respectively, whereas NIST, NCSC and BSI recommend AES-128 and SHA-256.

**Winternitz OTS.** WOTS is parameterized by two values:

- the Winternitz parameter  $w$ , being a power of 2.
- a  $n$ -bit hash function  $H$  such that  $n = vw$

To generate an OTS key pair, one randomly generates  $v$   $n$ -bit secret keys  $sk_0, \dots, sk_{v-1}$  and derives the corresponding public keys using an hash chain of length  $2^w - 1$  (*i.e.*,  $pk_i = H^{2^w-1}(sk_i)$ ). To sign a message  $m$ , a checksum over  $m$  is appended to it before hashing. The  $n$ -bit output is then divided into  $v$   $w$ -bit chunks  $c_0, \dots, c_{v-1}$  and the signature consists of  $\sigma = \sigma_0, \dots, \sigma_{v-1}$  where  $\sigma_i = H^{c_i}(sk_i)$ . To verify a signature, one checks that  $H^{w-1-c_i}(\sigma_i) = pk_i$  for  $i \in \{0, \dots, v-1\}$ .

**Merkle tree.** To build a many-time signature scheme from WOTS, one can combine multiple key pairs with a binary tree where each node is the hash of its children, commonly referred to as Merkle tree. For a height parameter  $h$ , such a tree is built from  $2^h$  leaves  $l_0, \dots, l_{2^h-1}$ , each being the hash of a WOTS public key (*i.e.*,  $l_i = H(pk_{i_0}, \dots, pk_{i_{v-1}})$ ). The root constitutes the many-time public key and commits to all OTS public keys. Note that to reduce memory requirements in practice, it is recommended to generate the WOTS secret keys using a pseudorandom function (PRF) rather than using a random number generator [HBG<sup>+</sup>18]. To sign the  $i$ th message, the signer uses the  $i$ th OTS secret key and includes the Merkle path of the corresponding public key in the signature. To verify the signature, the verifier computes the public key from WOTS signature and then, thanks to the Merkle path, verifies that its digest is indeed the leaf at position  $i$ . This introduces the concept statefulness: because security depends on the unique usage of each OTS key pair, it is crucial to keep track of which keys have already been used.

## 2.2 SNARK-based aggregation

The idea behind SNARK-based aggregation is for an aggregator to turn individual signatures, possibly over different messages, into a SNARK proof attesting their validity. Note that this principle can be used to thresholdize a signature scheme: given a  $k$ -of- $n$  setting, the aggregator can generate a proof attesting that it verified  $k$  distinct signatures over the same message and that signers are part of the quorum. A valuable feature of this approach is its non-interactiveness: the aggregator only needs to collect individual signatures in order to compute the proof, without any additional communication. Combining such a construction with hash-based signatures has been first explored by Khaburzaniya *et al.* [KCLM22], using WOTS with 1-bit chunks (instantiated with the Rescue-Prime hash function) along with STARKs. To complement this research, the work from Drake *et al.* [DKKW25] does not focus on a specific hash-based signature scheme but explores a variety of tradeoffs by introducing a generalized variant of XMSS [BDH11] and providing security proofs that hold for all its instantiations. Notably, their security proofs do not model hash functions as random oracles and rely on standard model properties instead, such as preimage/collision resistance, providing concrete security targets.

## 3 Towards threshold XMSS

The downside of building a threshold hash-based signature by leveraging a SNARK system as mentioned above is that the aggregation of threshold signatures would not be straightforward (since threshold signatures are proofs instead of raw hash-based signatures). In the case of the Beacon chain where threshold signatures occur *before* aggregation duties, it is imperative for distributed validator middlewares to output signatures that can be aggregated according to the protocol. Therefore, this section focuses on constructions which lead to threshold Winternitz signatures that are indistinguishable from non-threshold ones, as in BLS.

### 3.1 Distributed hash-based signatures with Boolean shares.

Distributed variants of hash-based signatures, including XMSS, have been explored by Kelsey, Lang and Lucks in [KLL22] where they introduce  $n$ -of- $n$  and  $k$ -of- $n$  threshold signature schemes which rely on Boolean shares. For the  $n$ -of- $n$  setting, a trusted dealer starts from an existing Merkle tree and splits each WOTS secret key  $sk_i$  by generating  $n$  random values  $r_i^0, \dots, r_i^{n-1}$  to compute  $r_i^h = r_i^0 \oplus r_i^1 \oplus \dots \oplus r_i^{n-1} \oplus sk_i$ . This introduces an additional party called the *helper* whose role is to store and provide the relevant helper shares whenever required. That way, to produce a WOTS using for  $sk_i$ , each party can sign independently using its Boolean key share assuming the aggregator has access to the helper share  $r_i^h$ . Note that it has to be done for each component of the secret key: assuming a WOTS scheme to sign  $n = vw$ -bit messages, it means that each WOTS key requires  $v2^w - 1$  helper shares. Furthermore, the trusted dealer also needs to provide the helper with shares for each Merkle paths, leading to high memory requirement for the helper overall. To minimize memory usage for the parties, the key shares are actually generated pseudorandomly using a PRF as detailed in Algorithm 2. To turn their  $n$ -of- $n$  scheme into a  $k$ -of- $n$  threshold scheme, they propose to instantiate a Merkle tree that contains keys for all possible  $\binom{n}{k}$  quorums. Beyond complexity, this increases the height of the Merkle tree and hence the signature size as well as the memory requirements for the helper. Because their designs are incompatible with distributed key generation (DKG) methods, we investigate other solutions instead.

### 3.2 Leveraging secret-sharing

Another approach is to leverage a secret-sharing scheme to split WOTS secret keys into shares distributed among participants. However, it requires to jointly compute all hash function calls in a multi-party computation (MPC) setting. This can be very challenging in practice, especially in low latency scenarios such as performing validator duties on Ethereum, as the time to produce a threshold signature may largely exceed the requirements (see *e.g.* the work from Cozzo and Smart [CS19] which estimates around 85 minutes to compute a threshold SPHINCS+ signature with SHA-3 as the underlying hash function). A possible workaround could be to evaluate the hash function calls in a distributed manner during key generation, so that each party can store a share for each component of each WOTS secret key. Note that this would require to be the done for all Merkle paths as well. Assuming a Merkle tree instantiated with a 256-bit hash function and  $2^{18}$  WOTS keys with a parameter  $w = 4$ , that would lead to a memory requirement of  $\approx 4.5$  gigabytes. A comprehensive performance analysis of MPC hash functions is crucial to assess what time-memory tradeoffs (*i.e.*, hashing WOTS secret key components on-the-fly instead of precomputing the value for each chunk) would be viable in practice.

## 4 Hash functions over MPC

Traditional hash functions (*e.g.*, SHA3) operate over binary fields for computational efficiency on a wide range of platforms. On the other hand, they lead to poor performance when employed within advanced cryptographic protocols such as MPC. This is mainly due to the fact that traditional schemes are designed to minimize their overall gate count without minimizing specifically nonlinear gates, which require communication between the parties in an MPC setting. Their inefficiency is such that it can constitute the bottleneck in MPC protocols, as highlighted by an attempt to thresholdize PQC signatures schemes [CS19]. In response, new primitives with design constraints finely tuned for advanced cryptographic protocols have emerged, known as *arithmetization-oriented* primitives.

Input parameters:

- Merkle tree built out of a  $n$ -bit hash function  $H$  and  $2^h$  WOTS secret keys  $sk_0, \dots, sk_{2^h-1}$  to sign  $n = vw$ -bit messages (*i.e.*,  $sk_i = (sk_{i,0}, \dots, sk_{i,v-1})$ ).
- A pseudorandom function  $PRF_K(x, l)$  parametrized by a  $k$ -bit key  $K$  which takes as input a seed  $x$  along with the output bit length  $l$ .
- A set of distributed parties  $\mathcal{P}$ .

Output parameters:

- Secret keys  $key_p$  for each party  $p \in \mathcal{P}$ .
- Helper shares  $sk_{i,j}^h$  and  $path_i^h$  for  $i \in \{0, \dots, 2^h - 1\}$  and  $j \in \{0, \dots, v - 1\}$ .

```

// picks secrets at random for each party
foreach  $p \in \mathcal{P}$  do
    |  $key_p \xleftarrow{\$} \{0, 1\}^k$ 
end foreach

// builds Merkle path helper shares
for  $i = 0$  to  $2^h - 1$  do
    |  $path_i^h \leftarrow path_i$ 
    | foreach  $p \in \mathcal{P}$  do
    | |  $path_i^h \leftarrow path_i^h \oplus PRF_{key_p}((domain_{path}, i), nh)$ 
    | end foreach
end for

// builds WOTS key helper shares
for  $i = 0$  to  $2^h - 1$  do
    | for  $j = 0$  to  $v - 1$  do
    | | for  $c = 0$  to  $2^w - 1$  do
    | | |  $sk_{i,j}^h[c] \leftarrow H^c(sk_{i,j})$ 
    | | | foreach  $p \in \mathcal{P}$  do
    | | | |  $sk_{i,j}^h[c] \leftarrow sk_{i,j}^h[c] \oplus PRF_{key_p}((domain_{key}, i, j, c), n)$ 
    | | | end foreach
    | | end for
    | end for
end for

```

**Algorithm 2:** Split a Merkle tree of WOTS keys into distributed key shares for  $n$ -of- $n$  signatures, according to [KLL22].

## 4.1 Beaver triples

## 4.2 MPC-friendly hash functions

Poseidon

Hydra

## References

- [BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 117–129, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 514–532, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [CS19] Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: Threshold Post-quantum Signatures. In Martin Albrecht, editor, *Cryptography and Coding*, pages 128–153, Cham, 2019. Springer International Publishing. <https://eprint.iacr.org/2019/1060>.
- [DKKW25] Justin Drake, Dmitry Khovratovich, Mikhail Kudinov, and Benedikt Wagner. Hash-Based Multi-Signatures for Post-Quantum Ethereum. Cryptology ePrint Archive, Paper 2025/055, 2025.
- [HBG<sup>+</sup>18] A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, RFC Editor, May 2018.
- [KCLM22] Irakliy Khaburzaniya, Konstantinos Chalkias, Kevin Lewi, and Harjasleen Malvai. Aggregating and Thresholdizing Hash-based Signatures using STARKs. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS ’22, page 393–407, New York, NY, USA, 2022. Association for Computing Machinery.
- [KLL22] John Kelsey, Stefan Lucks, and Nathalie Lang. Coalition and Threshold Hash-Based Signatures. Cryptology ePrint Archive, Paper 2022/241, 2022.
- [Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Stanford, CA, USA, 1979.
- [oST24a] National Institute of Standards and Technology. Module-Lattice-Based Digital Signature Standard. Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 204, U.S. Department of Commerce, Washington, D.C., 2024.
- [oST24b] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 203, U.S. Department of Commerce, Washington, D.C., 2024.
- [oST24c] National Institute of Standards and Technology. Stateless Hash-Based Digital Signature Standard. Technical Report Federal Information Processing Standards Publications (FIPS PUBS) 205, U.S. Department of Commerce, Washington, D.C., 2024.