Towards Threshold Hash-Based Signatures for Post-Quantum Distributed Validators

Alexandre Adomnicăi

DV Labs, alexandre@dvlabs.tech

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 Ethereum foundation recently introduced hash-based signature schemes as post-quantum alternatives to BLS. In this research, we study the practical challenges of deploying such schemes in a distributed manner.

Keywords: MPC · Hash-based signatures

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 sollicited the submission of additional signatures schemes to expand its PQC signature portfolio[1]. 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 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 which lends itself to Distributed Validator Technology (DVT) [Fou23]. 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 DVT middleware solutions, such as `charon`[2], which operates in a totally transparent manner from a consensus client point of view. However, because BLS is based on elliptic curve

[1] https://csrc.nist.gov/projects/pqc-dig-sig
[2] https://github.com/ObolNetwork/charon

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 atlernatives to BLS [DKKW25] as part of its consensus layer redesign[3]. 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 would have considerable impacts on distributed validators solutions which currently rely on the homomorphic properties of BLS to leverage threshold signatures. 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.

Table 1: Properties comparison between BLS and hash-based signature (HBS) schemes.

	BLS	HBS
post-quantum secure	✗	✓
native aggregation support	✓	✗
non-interactive threshold signing	✓	✗
deterministic	✓	✗

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[4], 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.

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 $sk_0, \cdots, sk_{v-1}$ and derives the corresponding public keys using hash chains 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 $c_0, \cdots, c_{v-1}$ and the signature consists of $\sigma = \sigma_0, \cdots, \sigma_{v-1}$ where $\sigma_i = H^{c_i}(sk_i)$. To verify a signature, one checks that $H^{2^w-1-c_i}(\sigma_i) = pk_i$ for $i \in \{0, \cdots, v-1\}$.

[3]https://beamroadmap.vercel.app/

[4]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.

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 $l_0, \cdots, l_{2^h-1}$, each being the hash of a WOTS public key ($i.e.$, $l_i = H(pk_{i_0}, \cdots, pk_{i_{v-1}})$). 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+18]. To sign the ith message, the signer uses the ith 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 $\mathsf{sk_i}$ by generating n random values $\mathsf{r}_i^0, \cdots, \mathsf{r}_i^{n-1}$ to compute $\mathsf{r}_i^h = \mathsf{r}_i^0 \oplus \mathsf{r}_i^1 \oplus \cdots \oplus \mathsf{r}_i^{n-1} \oplus \mathsf{sk}_i$. 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 1. 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. Overall, this approach comes with several limitations from a DVT perspective. First, it is incompatible with distributed key generation (DKG) algorithms since a trusted dealer is required to split the key into multiple shares. While supporting DKG is not a necessary prerequisite for distributed validators, it is a valuable feature as it ensures that the private key is never known in its entirety by any single party. Second, and more importantly, the helper role contradicts with the nature of DVT by introducing a single point of failure which affects decentralization. Therefore, we investigate alternative solutions that could overcome these weaknesses.

3.2 Leveraging secret sharing

Another approach is to leverage a linear 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 store all secret keys calculated during key generation, so that it is possible to sign messages efficiently without any online MPC calculation. However, this could lead to unrealistic memory usage as it requires to precompute secret keys for every possible chunk value, and this for all tree leafs. Since tree nodes are not considered secret material thanks to the preimage resistance of the underlying hash function, a more pragmatic approach would be to only store the plain (*i.e.*, non-shared) leafs value so that signers will be able to calculate Merkle paths in a non-distributed manner when generating signatures. Nevertheless, in the case of DKGs, this still requires computing all hash function calls over MPC at key generation time. A comprehensive performance analysis of MPC hash functions is necessary to evaluate the timing constraints of DKG setups and to identify practical time-memory tradeoffs for signature generation.

4 Hash functions over MPC

Traditional hash functions such as SHA3 operate over binary fields to enable efficient implementations in both hardware and software on a wide range of platforms. However, 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[5] which require communication between parties in an MPC setting, unlike linear gates that can be computed locally. The overload induced by these communications is such

[5]They are actually symmetric designs that aim at minimizing the number of nonlinear gates for efficient software masked implementations against side-channel attacks, see for instance [GLSV14].

```
Input parameters:
  • Merkle tree built out of a $n$-bit hash function $H$ and $2^h$ WOTS secret keys
    $\mathsf{sk}_0, \cdots, \mathsf{sk}_{2^h-1}$ to sign $n = vw$-bit messages ($i.e.$, $\mathsf{sk}_i = (\mathsf{sk}_{i,0}, \cdots, \mathsf{sk}_{i,v-1})$).
  • A pseudorandom function $\mathsf{PRF}_K(x, l)$ 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 $\mathsf{key}_p$ for each party $p \in \mathcal{P}$.
  • Helper shares $\mathsf{sk}^h_{i,j}$ and $\mathsf{path}^h_i$ for $i \in \{0, \cdots, 2^h - 1\}$ and $j \in \{0, \cdots, v - 1\}$.

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

// builds Merkle path helper shares
for $i = 0$ to $2^h - 1$ do
    $\mathsf{path}^h_i \leftarrow \mathsf{path}_i$
    foreach $p \in \mathcal{P}$ do
        $\mathsf{path}^h_i \leftarrow \mathsf{path}^h_i \oplus \mathsf{PRF}_{\mathsf{key}_p}\big((\mathsf{domain}_{\mathsf{path}}, i), nh\big)$
    end foreach
end for

// builds WOTS key helper shares
for $i = 0$ to $2^h - 1$ do                                  // for each WOTS secret key
    for $j = 0$ to $v - 1$ do                                // for each key component
        for $c = 0$ to $2^w - 1$ do                          // for each w-bit chunk
            $\mathsf{sk}^h_{i,j}[c] \leftarrow H^c(\mathsf{sk}_{i,j})$
            foreach $p \in \mathcal{P}$ do
                $\mathsf{sk}^h_{i,j}[c] \leftarrow \mathsf{sk}^h_{i,j}[c] \oplus \mathsf{PRF}_{\mathsf{key}_p}\big((\mathsf{domain}_{\mathsf{key}}, i, j, c), n\big)$
            end foreach
        end for
    end for
end for
```

Algorithm 1: Split a Merkle tree of WOTS keys into distributed key shares for n-of-n signatures, according to [KLL22].

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. They usually operate over $\mathbb{F}_p$ with p prime, making them natively compatible with linear secret sharing schemes, and rely on multiplications for nonlinear operations. Among them, Poseidon [GKR+21] has found its place into many Ethereum applications thanks to its efficiency in verifiable computing and its successor Poseidon2 [GKS23] is currently being considered for Ethereum protocols that rely on zero-knowledge proofs[6].

4.1 The Poseidon2 family of hash functions

Overview. Poseidon2 is built upon the Poseidon2$^\pi$ permutation operating over $\mathbb{F}_p^t$ with $p > 2^{30}$ prime and $t \in \{2, 3, 4, 8, 12, 16, 20, 24\}$. The permutation is meant to be combined with either a compression function or a sponge construction to build a hash function. Poseidon2$^\pi$ is based on the HADES design strategy which makes a distinction between external and internal rounds. Internal rounds (also called partial rounds) apply the nonlinear layer to only a part of the state, usually a single element, whereas external rounds (also called full rounds) process all elements in the same way. More precisely, Poseidon2$^\pi$ processes an internal state $x = (x_0, \cdots, x_{t-1}) \in \mathbb{F}_p^t$ as follows:

$$\mathsf{Poseidon2}^\pi(x) = \mathcal{E}_{R_F-1} \circ \cdots \circ \mathcal{E}_{R_F/2} \circ \mathcal{I}_{R_P-1} \circ \cdots \circ \mathcal{I}_0 \circ \mathcal{E}_{R_F/2-1} \circ \cdots \circ \mathcal{E}_0(M_\mathcal{E} \cdot x)$$

where $\mathcal{E}$ and $\mathcal{I}$ refer to external and internal round functions iterated for R_F and R_P rounds, respectively. Note that a linear layer is applied before running the first external round, which differs from the original Poseidon$^\pi$ design. The external/full round function is defined by:

$$\mathcal{E}(x) = M_\mathcal{E} \cdot \left(\left(x_0 + c_0^{(i)}\right)^d, \cdots, \left(x_{t-1} + c_{t-1}^{(i)}\right)^d\right)$$

where $d \geq 3$ is the smallest integer such that $\gcd(d, p-1) = 1$, $M_\mathcal{E}$ is a $t \times t$ maximum distance separable (MDS) matrix and $c_j^{(i)}$ is the j-th round constant for the i-th external round. The internal/partial round function is defined by:

$$\mathcal{I}(x) = M_\mathcal{I} \cdot \left(\left(x_0 + \hat{c}_0^{(i)}\right)^d, x_1, \cdots, x_{t-1}\right)$$

where $d \geq 3$ as before, $M_\mathcal{I}$ is a $t \times t$ MDS matrix and $\hat{c}_0^{(i)}$ is the round constant for the i-th internal round.

Efficient instantiations for hash-based signatures over MPC. Since Poseidon2 is a generic construction, all instantiations do not provide the same level of MPC-friendliness. Because all operations except exponentiations can be computed locally in an MPC setting, one should aim to minimize the d parameter as it would reduce the number of multiplications. Because the number of exponentiations is directly determined by the t and $R = R_F + R_P$ parameters, it is natural to seek to minimize their values. However, at the hash function level, selecting the optimal parameters depends on the size of the input to be processed. For large inputs that require a sponge mode as the underlying construction, having a large rate would allow to absorb more data per permutation, and eventually leading to fewer calls and fewer exponentiations in the end. In the case of hash-based signatures, most hash calls process small inputs to compute either hash chains from secret keys or nodes in the Merkle tree, with the exception of leafs which are obtained by hashing multiple public

[6] https://www.poseidon-initiative.info/

Table 2: Poseidon2$^{\pi}$ parameters for 31-bit prime fields. The reported results assume that cube evaluations take a single online communication round using the technique from [GRR+16] which requires 2 precomputed triples (1 Beaver + 1 cube).

Prime	**Parameters**				**Sbox impl.**	**MPC metrics**	
	t	d	R_F	R_P		depth	triples
Mersenne31 ($2^{31}-1$)	16	5	8	14	$(x^2)^2 \cdot x$	66	426
BabyBear ($2^{31}-2^{27}+1$)	16	7	8	13	$(x^2)^3 \cdot x$	63	564
KoalaBear ($2^{31}-2^{24}+1$)	16	3	8	20	x^3	28	296

keys. This is why the generalized XMSS scheme from [DKKW25] instantiates Poseidon2 with the compression mode for chain and tree hashing, whereas it uses the sponge mode for leaf hashing. For hash-based signatures over MPC however, one can disregard the specific case of leaf and tree hashing: since all inputs are public it is possible to recombine the shared values together to run the calculations in a non-distributed manner. Therefore, the rest of this document focuses on hash chains using Poseidon2 with the compression mode and $t = 16$ over a 31-bit prime field for efficient SNARK-based aggregation, as instantiated in [DKKW25]. We considered three different 31-bit prime fields which allow for efficient arithmetic, namely Mersenne31, KoalaBear and BabyBear. To compare the MPC-friendliness of the corresponding Poseidon2 instantiations, we do not consider the number of multiplications per se but the multiplicative depth instead. Indeed, since parallel (*i.e.*, independent) multiplications can be processed within the same communication round, minimizing the multiplicative depth is key when latency is a concern, as in distributed systems. As reported in Table 2, even though the KoalaBear prime field leads to the instantiation with the highest number of rounds, it is nevertheless the most MPC-friendly[7] thanks to its minimal d value, both in terms of multiplicative depth and precomputed data. Therefore, throughout the rest of this document, we focus on instantiations over the KoalaBear prime field for optimal online performance.

4.2 MPC protocols

MPC protocols greatly differ based on their security, adversarial and network assumptions. This section aims at identifying the relevant properties to target in the case of distributed validators.

Adversarial structure. Let n denote the number of participating parties and let t denote a bound on the number of parties that may be corrupted. MPC protocols are usually designed to provide security in either the dishonest (*i.e.*, $t < n$), honest (*i.e.*, $t < n/2$) or two-thirds honest (*i.e.*, $t < n/3$) majority settings. While dishonest-majority protocols offer the strongest security guarantees, they come at a significant cost. Restricting the model to an honest majority, however, enables substantial performance improvements. Thus, it is crucial to determine whether a dishonest-majority setting is truly necessary to achieve optimal efficiency. Regarding distributed validators, it is important to note that they rely not only on a threshold signature scheme (TSS) but also on a consensus algorithm[8], which ensures that all parties agree on the same message to be signed. It is well known that, in an asynchronous network, Byzantine fault tolerance (BFT) can only be achieved if $t < n/3$ [PSL80]. While adopting a two-thirds honest majority for MPC-based

[7] KoalaBear and BabyBear primes also show advantages over Mersenne31 when it comes to SNARKS thanks to their two-adic multiplactive subgroups for Cooley-Tukey NTTs.

[8] https://github.com/ethereum/distributed-validator-specs

signature computation aligns well with BFT assumptions, a higher fault tolerance threshold can be achieved if crash fault tolerance (CFT) suffices instead of BFT. In this case, an honest majority would enable the system to tolerate more faulty parties. Overall, while a two-thirds honest majority might be acceptable, an honest majority is desirable.

Adversarial behaviour. In the MPC litterature, corrupted parties are either considered malicious/active if they behave arbitrarily (*i.e.*, they can deviate from the protocol) or semi-honest/passive if they follow the protocol but combine their respective information to learn more than they should be allowed to. To ensure general compatibility with BFT consensus protocols, we aim for malicious security. Still, protocols in the semi-honest model can be of interest for efficiency if CFT-only is acceptable.

Security assumption. The security of an MPC protocol can rely on different assumptions. It can be *computational* secure, meaning that it depends on the hardness of specific mathematical problems (*e.g.*, factoring large numbers), or it can be *information-theoretic* secure, meaning that it achieves security based on principles of information theory without relying on computational hardness. While information-theoretic security is appealing to compute post-quantum over MPC, practical deployments rely on computational security for other components (*e.g.*, PRNGs for efficient preprocessing or secure communication channels). Since our end goal is to compute a post-quantum signature scheme in an MPC fashion, information-theoretic secure protocols are appealing as they are not threatened by quantum computers[9].

5 Experimental benchmarks

5.1 Benchmarking conditions

The MP-SPDZ framework. Benchmarking MPC protocols is challenging not only due to their technical complexity but also because their efficiency depends on various factors such as the security model, the number of participants and network conditions. In recent years, various frameworks have been developed to push the boundaries of practical MPC by making it more accessible. Among them, MP-SPDZ [Kel20] is very popular and actively maintained in an open source manner[10]. Users can write programs in high-level Python code, which is then extensively optimized and compiled into bytecode for a virtual machine implemented in C++. The framework implements a wide variety of protocols for several different security models, for both protocols based on secret sharing and garbled circuits. For our benchmarks, we opted for the ATLAS [GLO+21] and malicious Shamir[11][LN17] protocols which provide passive and active security, respectively, in the honest majority setting. Note that while ATLAS also comes in a version which provides active security, it is not supported by MP-SPDZ at the time of writing[12].

Setup. Benchmarks were run on an ASUS UX3405 laptop equipped with a Core Ultra 7 155H CPU. We simulate a five-party setting over a network with 10ms latency and 200Mbps bandwidth, reflecting a distributed setup with optimistic networking conditions. The code to be compiled with MP-SPDZ, is available at `https://github.com/ObolNetwork/pqdv/blob/main/mpspdz/poseidon2.mpc`.

[9]Practical MPC deployments actually rely on computational security for other components (*e.g.*, PRNGs for efficient preprocessing or secure communication channels).
[10]`https://github.com/data61/MP-SPDZ`
[11]Note that the `malicious-shamir` protocol from MP-SPDZ relies on another protocol for the preprocessing phase, see `https://github.com/data61/MP-SPDZ/issues/386#issuecomment-935384621`.
[12]`https://github.com/data61/MP-SPDZ/issues/1578`

5.2 One-time signature

Hash chains over MPC. While Table 2 provides the multiplicative depth for various Poseidon2 instantiations, it is for a *single* permutation call only. In the case of WOTS, the overall depth actually depends on the hash chains' length (*i.e.*, the number of successive Poseidon2$^{\pi}$ calls). As detailed in Section 2.1, the length of each chain depends on the chunk value it signs and is therefore bounded by $2^w - 1$, where w refers to the Winternitz parameter which defines a trade-off between speed and signature size. Hence a greater value of w implies a smaller signature but slower speeds and vice-versa. While the encoding method does not impact the multiplicative depth as chunks can be processed in parallel, it still affects overall efficiency as a higher number of chunks leads to more preprocessed triples and increased network bandwidth usage. For our study, we considered the XMSS instances from [DKKW25] with the constant-sum encoding scheme as it provides the optimal rate for WOTS signatures [ZCY23] (*i.e.*, it minimizes the number of encoded chunks and hence the signature size in the end) whose corresponding MPC metrics are reported in Table 3.

Table 3: MPC metrics for XMSS signature generations when instantiated with Poseidon2 over a 31-bit prime field, using the target-sum encoding. The number of preprocessing triples are reported for both the worst case (*i.e.*, all chunks equal $2^w - 1$) and the average case.

Parameters		**MPC metrics**		
w	chunks	depth	triples (WC)	triples (AC)
1	155	28	45 880	22 940
2	78	84	69 264	34 632
4	39	420	173 160	86 580
8	20	7140	1 509 600	754 800

Benchmark results for OTS generation are reported in Table 4 for $w \in \{1, 2, 4\}$ where all chunks are processed in parallel and take advantage of the same communication rounds. As expected, the timings are highly dominated by the network conditions, especially on latency rather than bandwidth as the number of communication rounds constitute the bottleneck, and therefore are close to `network_latency` $\times$ #`com_rounds`. Although signing over MPC can be achieved in less than a second for the smallest Winternitz parameter and assuming a network latency of 10ms, a greater w value and less optimistic network conditions leads to multiple seconds. A potential workaround could be to precompute all hash chain intermediate values to avoid MPC calculations during the signature generation.

Time-memory tradeoffs. As mentioned in the previous section, another alternative would be to precompute all hash chain intermediate values, for each secret key and for every possible chunk. Therefore the memory requirement to store all hash chains' intermediate values, for each secret key and for every possible chunk, would be $2^w \cdot \mathsf{chunks}$ digests per one-time key. Assuming the target-sum encoding where digests are composed of 7 31-bit field elements, along with $w = 2$ it would mean $(78 \cdot 4 \cdot 7 \cdot 31)/8 = 8\,463$ bytes per one-time key. Depending on the Merkle tree height, this might lead to considerable memory requirement (*e.g.*, more than 8GiB for 2^{20} leafs). An in-between solution could be to not store *all* the intermediate values but only some of them, at different positions within the chain. For instance restricting the storage to the values at the i-th positions only where $i \pmod{w} = 0$ would allow to reduce the storage requirement by a factor of w, while requiring at most w Poseidon2$^{\pi}$ calls per chunks instead of $2^w - 1$.

Table 4: Benchmarks of a random message signature generation over MPC using the MP-SPDZ framework, in a five-party setting. Only hash chains are considered (*i.e.*, calculations that do not require MPC are not taken into account). Results were obtained by averaging the timing results over 10 executions for 3 different network delays (10, 50 and 100ms) to reflect various network conditions.

Winternitz parameter	**Protocol**	**Offline phase**					**Online phase**					**Combined**			
		Prec.	Time (s)			Data (MB)	Com. rounds	Time (s)			Data (MB)	Time (s)			Data (MB)
			$d = 10$	$d = 50$	$d = 100$			$d = 10$	$d = 50$	$d = 100$		$d = 10$	$d = 50$	$d = 100$	
1	ATLAS	11 396	0.21	0.93	1.83	0.28	65	0.72	3.29	6.49	0.54	0.93	4.22	8.32	0.76
	malicious Shamir	22 792	0.40	1.72	3.37	2.53	30	0.38	1.62	3.20	1.09	0.78	3.34	6.57	3.62
2	ATLAS	17 316	0.21	0.93	1.83	0.34	177	1.93	8.96	17.76	0.78	2.14	9.89	18.10	1.12
	malicious Shamir	34 632	0.46	1.95	3.77	3.82	86	1.01	4.44	8.76	1.66	1.47	6.39	12.35	5.48
4	ATLAS	43 216	0.39	1.72	3.37	0.84	855	9.11	43.33	86.06	1.96	9.50	45.05	89.43	2.80
	malicious Shamir	86 432	1.01	4.12	8.00	9.47	422	4.63	21.59	42.73	4.15	4.64	25.71	50.73	13.62

Table 5: Benchmarks of an XMSS DKG using the MP-SPDZ framework, in a five-party setting. Only hash chains are considered (*i.e.*, calculations that do not require MPC are not taken into account).

Winternitz parameter	**WOTS keys**	**Protocol**	**Offline phase**			**Online phase**			**Combined**	
			Prec.	Time (s)	Data (GB)	Com. rounds	Time (s)	Data (GB)	Time (s)	Data (GB)
1	2^{10}	ATLAS	2.34×10^7	153	0.42	6.40×10^4	717	1.05	870	1.47
		malicious Shamir	4.68×10^7	500	5.13	2.96×10^4	410	2.25	910	7.39
	2^{12}	ATLAS	9.39×10^7	623	1.70	2.56×10^5	2 934	4.21	3 557	5.91
		malicious Shamir	18.79×10^7	2 163	20.55	1.18×10^5	1 830	9.02	3 993	29.57
	2^{14}	ATLAS	3.75×10^8	2 451	6.81	10.25×10^5	11 505	16.83	13 957	23.65
		malicious Shamir	7.51×10^8	8 012	82.19	4.75×10^5	6 576	36.08	14 588	118.28
2	2^{10}	ATLAS	3.56×10^7	225	0.65	18.18×10^4	1 971	1 595	2 197	2.24
		malicious Shamir	7.02×10^7	817	7.78	8.74×10^4	1 147	3.42	1 964	11.2
	2^{12}	ATLAS	1.42×10^8	938	2.57	7.24×10^5	8 096	6.35	9 034	8.92
		malicious Shamir	2.84×10^8	3 140	31.02	3.48×10^5	4 408	13.61	7 548	44.64
	2^{14}	ATLAS	5.67×10^8	3 594	10.28	2.89×10^6	31 489	25.42	35 082	35.70
		malicious Shamir	11.32×10^8	12 115	124.1	1.39×10^6	17 034	54.47	29 149	178.56

5.3 Distributed key generation

Benchmark settings. A DKG inconditionally requires to compute $2^w - 1$ hash chains over MPC, for each WOTS secret key and every chunk value, therefore resulting in a way more intensive process compared to a signature generation. To provide concrete numbers, we run benchmarks for $w \in \{1, 2\}$ along with the target-sum encoding. Due to long running time, we only considered a network latency of 10ms along with three different number of tree leafs: $2^{10}, 2^{12}$ and 2^{14}. Although they do not reflect realistic deployment parameters due to their small values, it allows to conjecture on results for a larger number of WOTS keys. Contrary to the signature generation benchmark where parallelization capabilities were fully exploited (*i.e.*, a single communication round handles all chunks simultaneously), the DKG benchmark only partially leverages the processing of independent chunks. More precisely, it does not leverage parallelization capabilities at the WOTS keys' level (*i.e.*, a single communication round handles all chunks simultaneously, but for a single WOTS key) leading to more communication rounds. This choice is justified by the fact that the MP-SPDZ framework was not able to unroll all loops due to programs' complexity (without mentioning that a fully parallelized implementation is not realistic in terms of bandwidth requirements).

Results interpretation. As reported in Table 5, the timings are still largely dominated by the network conditions, especially latency. Still, while bandwidth was not a major concern for signature generation, it becomes significant here as the amount of data exchanged between parties can exceed a hundred gigabytes. This gives an advantage to the ATLAS protocol which reduces the amount of precomputed data in the offline phase by half, lowering the amount of data transferred by an order of magnitude compared to malicious Shamir (although it doubles the number of online communication rounds while providing only passive security). While the distinction between the offline and online phases is less obvious in DKG compared to signature generation, it is still worth distinguishing between the two. Indeed, since DKG is run only once, it may be reasonable to assume that a trusted third party provides all parties with the preprocessed data so that they can directly proceed to the online phase.

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.

[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.

[Fou23] Ethereum Foundation. Distributed validator technology. https://ethereum.org/en/staking/dvt/, 2023. Accessed: 2025-02-10.

[GKR$^+$21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535. USENIX Association, August 2021.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A Faster Version of the Poseidon Hash Function. In *Progress in Cryptology - AFRICACRYPT 2023: 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19–21, 2023, Proceedings*, page 177–203, Berlin, Heidelberg, 2023. Springer-Verlag.

[GLO$^+$21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: Efficient and Scalable MPC in the Honest Majority Setting. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 244–274, Cham, 2021. Springer International Publishing.

[GLSV14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *FSE*, pages 18–37. Springer, 2014.

[GRR$^+$16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-Friendly Symmetric Key Primitives. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 430–443, New York, NY, USA, 2016. Association for Computing Machinery.

[HBG$^+$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.

[Kel20] Marcel Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1575–1590, New York, NY, USA, 2020. 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.

[LN17] Yehuda Lindell and Ariel Nof. A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 259–276, New York, NY, USA, 2017. Association for Computing Machinery.

[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.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2):228–234, April 1980.

[ZCY23] Kaiyi Zhang, Hongrui Cui, and Yu Yu. Revisiting the Constant-Sum Winternitz One-Time Signature with Applications to SPHINCS+ and XMSS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 455–483, Cham, 2023. Springer Nature Switzerland.